2017

Modeling Faceted Browsing with Category Theory for Reuse and Interoperability

Daniel R. Harris
University of Kentucky, daniel.harris@uky.edu
Author ORCID Identifier:
http://orcid.org/0000-0001-9139-3433
Digital Object Identifier: https://doi.org/10.13023/ETD.2017.138

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Daniel R. Harris, Student

Dr. Jerzy W. Jaromczyk, Major Professor

Dr. Miroslaw Truszczynksi, Director of Graduate Studies

Modeling Faceted Browsing with Category Theory for Reuse and Interoperability

DISSERTATION

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the College of Engineering at the University of Kentucky

By
Daniel R. Harris
Lexington, Kentucky

Director: Dr. Jerzy W. Jaromczyk
Lexington, Kentucky 2017

ABSTRACT OF DISSERTATION

Modeling Faceted Browsing with Category Theory
for Reuse and Interoperability

Faceted browsing (also called faceted search or faceted navigation) is an exploratory search model where facets assist in the interactive navigation of search results. Facets are attributes that have been assigned to describe resources being explored; a faceted taxonomy is a collection of facets provided by the interface and is often organized as sets, hierarchies, or graphs. Faceted browsing has become ubiquitous with modern digital libraries and online search engines, yet the process is still difficult to abstractly model in a manner that supports the development of interoperable and reusable interfaces. We propose category theory as a theoretical foundation for faceted browsing and demonstrate how the interactive process can be mathematically abstracted in order to support the development of reusable and interoperable faceted systems.

Existing efforts in facet modeling are based upon set theory, formal concept analysis, and light-weight ontologies, but in many regards they are implementations of faceted browsing rather than a specification of the basic, underlying structures and interactions. We will demonstrate that category theory allows us to specify faceted objects and study the relationships and interactions within a faceted browsing system. Resulting implementations can then be constructed through a category-theoretic lens using these models, allowing abstract comparison and communication that naturally support interoperability and reuse.

In this context, reuse and interoperability are at two levels: between discrete systems and within a single system. Our model works at both levels by leveraging category theory as a common language for representation and computation. We will establish facets and faceted taxonomies as categories and will demonstrate how the computational elements of category theory, including products, merges, pushouts, and pullbacks, extend the usefulness of our model. More specifically, we demonstrate that categorical constructions such as the pullback and pushout operations can help organize and reorganize facets; these operations in particular can produce faceted views containing relationships not found in the original source taxonomy. We show how our category-theoretic model of facets relates to database schemas and discuss how this relationship assists in implementing the abstractions presented.

We give examples of interactive interfaces from the biomedical domain to help illustrate how our abstractions relate to real-world requirements while enabling systematic reuse and interoperability. We introduce DELVE (Document ExpLoration and Visualization Engine), our framework for developing interactive visualizations as modular Web-applications in order to assist researchers with exploratory literature search. We show how facets relate to and control visualizations; we give three examples of text visualizations that either contain or interact with facets. We show how each of these visualizations can be represented with our model and demonstrate how our model directly informs implementation.

With our general framework for communicating consistently about facets at a high level of abstraction, we enable the construction of interoperable interfaces and enable the intelligent reuse of both existing and future efforts.

KEYWORDS: faceted browsing, category theory, abstract models, interactive systems, reusability, interoperability

Author's signature: Daniel R. Harris

Date: April 29, 2017

Modeling Faceted Browsing with Category Theory
for Reuse and Interoperability

By
Daniel R. Harris

Director of Dissertation: Jerzy W. Jaromczyk

Director of Graduate Studies: Miroslaw Truszczynski

Date: April 29, 2017

Dedicated to my wife, Cherron, whose love and support seems to have no end.

ACKNOWLEDGMENTS

This work is the result of a long sequence of events that ultimately put me in a position where I was trusted to independently develop novel and interesting solutions to a very practical research problem. I would like to thank a number of individuals and organizations for their mentorship, participation, and support during the development of this dissertation.

I would like to especially thank my advisor and committee chair, Dr. Jerzy Jaromczyk, for providing guidance, support, and an endlessly unique perspective that has without doubt helped shape my own ability to solve problems and conduct research. I would like to express my appreciation and gratitude for having the luck and pleasure of being mentored by Dr. Jaromczyk as an undergraduate, master's, and Ph.D student.

I would like to thank my committee members, Todd Johnson, Ramakanth Kavuluru, Jinze Liu, and Chen Qian, for their helpful comments and feedback, which have improved the quality of this work. I would like to also extend thanks to Dr. Kenneth Kubota who agreed to participate as an outside examiner. I would like to particularly thank Dr. Todd Johnson for organizing and evangelizing the DELVE project at both UK and UT/Houston. Similarly, I would like to thank Ramakanth Kavuluru for his guidance and participation with DELVE and other collaborations.

Although my contribution is clearly written for a general computer science audience, my graduate student life involved maintaining a balance between my role as a computer science graduate student and my role as a full-time staff member for the biomedical informatics core of the Center for Clinical and Translational Science (CCTS). It was at the the CCTS where I helped bootstrap the DELVE project from which I carved out a focused research question for my dissertation. I would like to

thank Drs. Todd Johnson and Jeffery Talbert for placing me in a role that actively engages and benefits from my computer science background.

I would like to thank leadership at the CCTS for financially supporting the development and publishing of material related to this dissertation through grant numbers UL1TR001998 and UL1TR000117. Additionally, I would like to thank the Institute of Pharmaceutical Outcomes and Policy and Dr. Jeffery Talbert for sponsoring travel to IEEE IRI 2016. I would like to thank the anonymous reviewers of my conference and journal articles, especially those individuals who were clearly experts on category theory: I have no idea who you are, but your suggestions were invaluable.

I am in debt to my family for their unwavering support and understanding. My wife, Cherron, has unconditionally supported me through the long days and nights required to complete this work. My sons, Simon and Atticus, have also remained patient and supportive; I hope this one day inspires them to excel and answer the call of their true passions in life, whatever they may ultimately be.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

Faceted classification is the process of assigning facets to resources in a way that enables intelligent exploratory search aided by an interactive faceted taxonomy [1]. Facets are the individual elements of a faceted taxonomy and are simply attributes known to describe an object being cataloged; these collections of facets are often organized as sets, hierarchies, lattices, or graphs. Facets are usually shown alongside a list of other related, relevant facets which aid in interactive filtering and expansion of search results [2]. Exploratory search using a faceted taxonomy is often called faceted browsing (or faceted navigation or faceted search) [3] and is commonly utilized in digital libraries and online search engines.

Facet models formalize faceted data representations and the subsequent interactive operations for exploratory search tasks. Wei et al. observed three major theoretical foundations behind current research of facet models: set theory, formal concept analysis, and lightweight ontologies [1]. In this contribution, we demonstrate that category theory can act as a theoretical foundation for faceted browsing that encourages reuse and interoperability by uniting different facet models together under a common framework [4, 5]. We also establish facets and faceted taxonomies as categories and demonstrate how the computational elements of category theory, such as products, functors, pushouts, and pullbacks extend the utility of our model [4]. The usefulness of faceted browsing systems is well-established in the digital libraries research community [6, 7], but reuse and interoperability are typically not major design considerations [4]. Our goal is to create a rich environment for faceted browsing where reuse and interoperability are primary design considerations.

1.1 Motivation

The motivation for choosing category theory began when designing the next phase of DELVE (Document ExpLoration and Visualization Engine) [8], our framework for creating visualizations for browsing biomedical literature. Specifically, we encountered difficultly in modeling DELVE's ability to create numerous visualizations, which are either controlled by facets or contain faceted structures. Additionally, one visualization may impact another either by filtering or focusing. For example, how can one effectively represent a hierarchical tree of facets which is simultaneously browsable and capable of spawning faceted graphs containing interactive, linked nodes? A

model of faceted browsing that is capable of representing faceted taxonomies generically would enable the quick creation of interoperable faceted components within an interface and enable their reuse in either alternate parts of the interface or in a different interface altogether. Although this abstraction is possible with set theory, the notation quickly becomes cluttered and error-prone. If set theory was used to address the question in the example above, individual sets would be necessary for each component and additional index sets would be needed for every type of relationship that maps one set to another set; linking nodes becomes an exercise in managing set indices and the sets necessary for facilitating interactivity. Additionally, it is difficult to incorporate existing work on faceted browsing due to the vast variety of models and implementations. A modeling methodology that is capable of operating at a high level of abstraction is necessary.

Some faceted systems, such as hierarchical faceted categories [3, 9], are implemented without a true theoretical foundation [1]; in this context, categories refer to how facets categorically index items and are not related to category theory. In general, category theory aims to represent objects and relations at their most intrinsic, abstract level and is appropriate for modeling problems in the sciences [10], including computer science [11]. The volume of existing work for faceted browsing systems lends itself to a higher degree of abstraction, where existing works can become interoperable and reusable in new research settings; we integrate existing facet models into our model in Chapter 4. In Chapter 3, we will demonstrate that category theory is an appropriate framework for developing such abstractions by establishing facets and faceted taxonomies as categories in the mathematical sense.

1.2 Preliminary Comments on Category Theory

On utilizing category theory, Benjamin C. Pierce writes:

"One controversial point in any discussion of the applicability of category theory to computer science is *how much* of category theory people are interested in using. Some authors [for example, Reynolds] use category theory simply as a powerful and uniform notational framework for masses of complicated but relatively elementary detail. On the other hand, Dybjer cites papers by Lehman, Goguen, and Burstall where deep theorems of category theory are applied to computational situations" [12].

This thesis is partly premised on the notion that category theory provides a *powerful and uniform notational framework* for modeling faceted browsing. The challenge is then identifying what elements of category theory are appropriate and useful in ex-

tending the model's categorical foundation. We further assert that modeling faceted browsing with category theory enables reusability at two distinct levels: within a faceted browsing system and across faceted browsing systems.

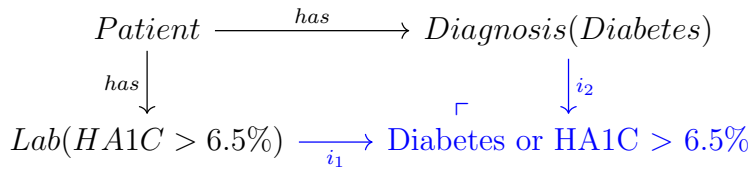
1.3 Thesis structure and contributions

Chapter 1 contains a brief introduction, a discussion regarding the motivation for our contribution, and an outline of the structure for the thesis that details each chapter's contribution. Chapter 2 includes background knowledge for both faceted browsing and category theory to aid the reader in comprehending the proposed category-theoretic model presented in Chapter 3.

Chapter 3 introduces facets and faceted taxonomies as formal categories, which intuitively can be thought of as a collection of objects and relationships between those objects where identity and associative composition functions are well-defined. The objects in question are collections of pointers to resources; the relationships between objects describe the faceted structure surrounding each resource. Once the model is established, we can leverage the abstractions for computation and show how products, merge operations, and pushout and pullback operations can manipulate facets in a meaningful manner. To preview utility, consider a scenario where patients are the resources being explored and they are described by their diagnoses, their medications, their procedures, and their lab values. If we wish to target patients who suffer from diabetes, we could select the diagnosis code corresponding to diabetes, but because of known medical coding quality issues, it would not identify every patient with diabetes. It is a known issue that not every individual with diabetes will be assigned a diagnostic code formally indicating that they have diabetes; to combat this shortcoming of the data, researchers often search for criteria that would imply that the patient has diabetes, such as the presence of medications specific to diabetes or the presence of lab values that would indicate that this person likely has diabetes [13]. For example, we can consider searching for patients who either have been assigned a diagnostic code for diabetes or have a specific lab value:

$$\begin{array}{ccc} Patient & \xrightarrow{has} & Diagnosis(Diabetes) \\ {\scriptstyle has}\downarrow & & \\ Lab(HA1C > 6.5\%) & & \end{array}$$

The pushout, indicated in blue below, can capture patients that meet either criteria:



At this point, the new facet can be referenced and used as part of the interface as if it were in the original taxonomy. We expand upon this example in Section 3.7 and indicate how pushouts can help construct faceted views of the underlying data, where computed facets that do not necessarily exist can be used to address short-comings in the original data (such as patients with diabetes being poorly coded by diagnosis codes).

In Chapter 4, we describe how our model supports reuse at two different levels: within a system and across systems. Within a system, the key benefit to modeling faceted taxonomies through category theory is that we can reuse the facets and reframe their relationships (or morphisms) to fit our needs; if we need to arrange the facets as graphs, we can easily do so. To encourage reuse across systems, the essential requirement for reusing components is that a common language exists that can bridge across components and enable communication and interactivity between the two. We give examples of how our model could model existing faceted solutions and unify different models together into our common framework; using a well-known facet model as an example, we also show how to reconcile incompatibilities between our model and existing models.

In Chapter 5, we give details on transitioning from creating reusable abstractions to creating reusable implementations. We show that our category-theoretic model can be mapped to a known category representing database schemas, originally introduced by Spivak [10]. Given that there exists a mapping between facet and schemas, we give details on how facets can be implemented within a relational database by introducing instances of categories. We give examples of biomedical facets and show how they are implemented after being modeled with our proposed abstractions. An additional benefit of instances is that we can use them to model interfaces that require multiple taxonomies and that their implementations are portable across faceted systems and may be reused.

In Chapter 6, we further our discussion on the different requirements of faceted browsing systems and show how interfaces that require multiple taxonomies can be abstracted and implemented. Because the needs of interactive interfaces vary, we also discuss two different strategies for supporting taxonomies derived from multiple heterogeneous terminologies: merging into a master taxonomy and maintaining multiple

instances. To motivate the use case of merging, we discuss the open-source project i2b2 (Integrating Biology Bench to Bedside) [14] as a hypothetical and prototypical example from the biomedical domain where merging is a necessity.

In Chapter 7, we discuss DELVE (Document ExpLoration and Visualization Engine) as an example of a faceted browsing system that uses multiple taxonomies independently without merging. DELVE is unique in that it contains visualizations that are either controlled by facets or contain facets. We detail how to abstract DELVE with our model by examining each component; we outline clouds, word trees, and phrase nets and show how they can be represented with our model of faceted browsing. The consistency that category theory provides in its language allows each visualization to communicate with each other and this interoperability is carried down to the implementation where each component can interact with and manipulate other visualizations. We highlight the path from abstraction to implementation with each visualization technique and emphasize how DELVE, as a framework, encourages each component to be reused.

1.4 Related Publications

This thesis contains material that was presented in part at the following conferences and journals:

1. **Daniel R. Harris** " Modeling Terminologies for Reusability in Faceted Systems." Advances in Intelligent Systems and Computing, pp 1-25. Springer, 2017.

2. **Daniel R. Harris**, Ramakanth Kavuluru, Jerzy W. Jaromczyk, Todd R. Johnson. "Rapid and Reusable Text Visualization and Exploration Development with DELVE". In Proceedings of the 2017 American Informatics Association (AMIA) Joint Summits (CRI), pp. 1-10. AMIA, 2017.

3. **Daniel R. Harris** "Modeling integration and reuse of heterogeneous terminologies in faceted systems." In Proceedings of the 2016 IEEE International Conference on Information Reuse and Integration (IRI), pp. 58-66. IEEE, 2016. `https://doi.org/10.1109/IRI.2016.16`

4. **Daniel R. Harris** "Foundations of reusable and interoperable facet models using category theory." Information Systems Frontiers, vol. 18, no. 5, pp. 953-965. Springer, 2016. `https://doi.org/10.1007/s10796-016-9658-6`

5. **Daniel R. Harris** "Modeling reusable and interoperable faceted browsing systems with category theory." In Proceedings of the 2015 IEEE International Conference on Information Reuse and Integration (IRI), pp. 388-395. IEEE, 2015. https://doi.org/10.1109/IRI.2015.65

6. **Daniel R. Harris** "Modeling faceted browsing with category theory to support interoperability and reuse." In Proceedings of the 15th ACM/IEEE-CE on Joint Conference on Digital Libraries, pp. 275-276. ACM, 2015. https://doi.org/10.1145/2756406.2756972

Chapter 2 Background

Our contributions intersect the worlds of faceted browsing and category theory. We first discuss faceted browsing and indicate where our work falls within the scope of faceted browsing research. We then discuss category theory and introduce concepts that are necessary to understand the category-theoretic model that we propose in Chapter 3.

2.1 Faceted Browsing Research

In the 1930s, S. R. Ranganathan developed colon classification, the first example of faceted classification [1]. In colon classification, library materials are sorted using five facets (personality, matter, energy, space, and time) [1]. As a direct consequence of faceted classification, faceted browsing became possible.

Faceted browsing (also called faceted search or faceted navigation) is an exploratory search model, where facets assist in the navigation of search results [3]. Facets are simply attributes attached to the actual objects being explored. An example of a facet attached to a book could be its genre or publication date. In a typical faceted browsing system, a user is shown search results alongside a list of related, relevant facets, allowing interactive filtering and expansion of results [2]. A faceted taxonomy is the collection of facets provided by the interface and is often organized as sets, hierarchies, or graphs.

A survey of faceted browsing research is summarized in Table 2.1; Wei developed four main facets of faceted browsing research: facet models, key technologies, evaluation metrics, and faceted search systems [1].

Table 2.1: Abbreviated Summary of Faceted Research

Facet Models	**Key Technologies**	**Evaluation Metrics**
Theoretical basis	Facet extraction	Subjective metrics
Model structure	Hierarchy construction	Objective metrics
Interactivity	Facet ranking	Relevance metrics
...	...	...

Our contribution falls under the first type of faceted browsing research, highlighted in blue in Table 2.1. Research on facet models examines the formal representation of the faceted browsing process. Work on key technologies innovates the creation

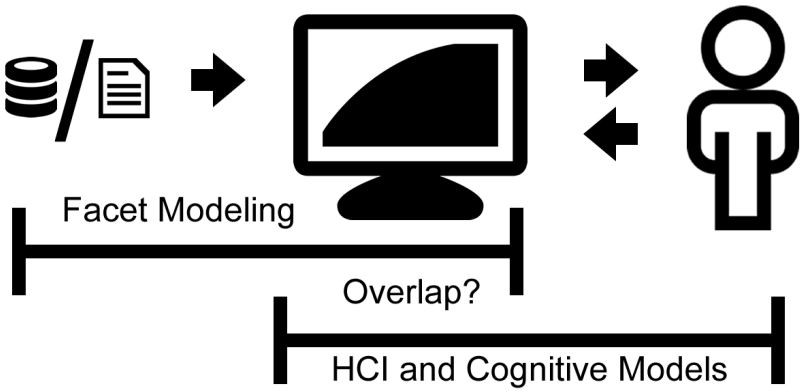


Figure 2.1: The scope of facet modeling vs human-computer interaction and cognitive models

and navigation of faceted browsing systems. Research on evaluation metrics offers techniques to evaluate the quality of facets generated or the system's ability to deliver relevant content. Not depicted in the table, the last main facet of current faceted browsing research, "faceted search systems", is dedicated to discussing novel faceted browsing tools.

2.1.1 Facet Models

Most efforts in facet modeling focus on the formal representation of faceted data and what interactive operations are made available through the model. This is significantly different than efforts modeling cognitive processes and human-computer interaction (HCI), yet overlaps between these two topics do exist, as illustrated in Figure 2.1. At the heart of the overlap is the interface itself: the facet model must map functionality to the interface and HCI/cognitive models must map communication and comprehension between the interface and the user. Unification of these models is possible as future work and is discussed in Chapter 8.

The design of a system's underlying model directly impacts the user's ability to filter, rank, and interact with the facets; in fact, some models contain no interactivity [1]. Wei et al. observed three major theoretical foundations behind current research of facet models: set theory, formal concept analysis, and lightweight ontologies. Facet modeling focuses on the formal representation of faceted data and the interactive consequences that follow when using that model.

In the next section, we give examples of faceted browsing systems from each type of theoretical foundation. Most of these systems were presented in a manner similar to that illustrated in Figure 2.2, where the abstract facet model is described,

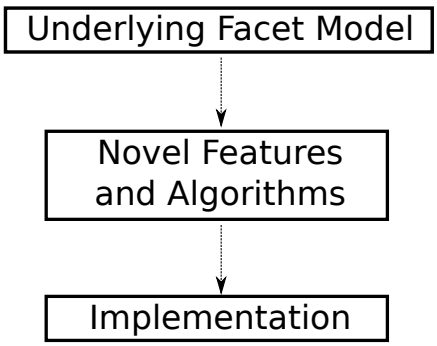


Figure 2.2: Common presentation of a faceted browsing system

followed by any novel features and algorithms. Implementation is typically discussed last, as it is least important in explaining why the interface is novel and interesting. This demonstrates an emphasis on the importance of the model to a system and helps justify why our modeling work should be instrumental for improving the development of faceted browsing systems.

2.2 Faceted Browsing Systems

Faceted browsing systems enable effective use of faceted taxonomies and facet classification, the process of assigning facets to the resources to be queried. The utility of faceted classification and faceted taxonomies is well-understood [3, 2, 6], even as a pivotal element to modern information retrieval [15]. Facet browsing is domain agnostic, but it is a natural fit for domains that have a rich history of ontological integration such as biomedicine [8, 16]. Faceted taxonomies can aid in the construction of information models [17] or aid in the construction of a larger ontology [18]. We focus on modeling faceted browsing in a way that enables the design of reusable and interoperable faceted taxonomies within the interface.

In order to fully understand our motivation, we must discuss existing faceted browsing models. When discussing these efforts, it is important to keep in mind that these models were constructed for a single system with a single faceted taxonomy; although each system was clearly an innovative and successful initiative, reusability and interoperability with future systems was not a priority or consideration discussed.

Additionally, basic knowledge of category theory is necessary to understand our model. We will introduce category theory in Section 2.3 using Spivak's definition of categories which was originally aimed toward the sciences audience [10].

2.2.1 Foundations of Facet Models

Of the three major foundations of facet modeling, set theory efforts tend to provide a model that explicitly includes interactive operations, such as filtering and ranking [1]. Formal concept analysis focuses on defining facets and faceted structures for knowledge [19] and has deep-seated roots in lattice theory in order to provide an organizational structure to faceted browsing. Light-weight ontologies provide an easy way to apply natural-language labels to concepts organized in an ontology [20], but do not explicitly model interactive operations.

Any implementation of faceted browsing, whether its foundation be grounded in set theory or lattice theory, could be abstracted into a category-theoretic framework as objects and relations. Because natural connections between category and set theory exist [21, 10], our work is most comparable to existing efforts in set theory. The facet modeling efforts that explicitly use set theory as a foundation differ in their core definitions and how they model filtering and ranking of facet objects: Dynamic Taxonomies [22] is a classic way of dynamically representing taxonomies with *is-a* relationships; Category Hierarchies are defined as connected, rooted directed acyclic graphs [23]; Generalized Formal Models [24] use entity-relationship diagrams to represent faceted hierarchies; FaSet [25] implements facets and queries as sets within relational databases. Each of these implementations has its own base definition of what it means to be a facet. In FaSet, a facet F is a set of items and if the system has multiple facets, they are disjoint: $F_a \cap F_b = \emptyset$ [25]. The model is then constructed axiomatically using the base definition of a facet.

Simply due to their shared abstract nature, our work is also similar to efforts based on formal concept analysis [19]. We focus on modeling faceted structures once a representation for knowledge has been chosen, including lattices from formal concept analysis; we do not compete with formal concept analysis, but rather enable its reuse by providing a model capable of representing it consistently with other faceted structures. In other words, a lattice can peacefully coexist and interact with simpler structures such as sets and hierarchies. Many systems contain only one faceted taxonomy, but systems like DELVE can contain multiple visualizations; these visualizations either contain or are controlled by independent faceted taxonomies that may or may not share the same structure.

An example would be an interface that initially contains a visualization of a simple dynamic hierarchy depicting basic *is-a* relationships; for a given node, the interface could interactively allow one to visualize more complex relationships that are stored in a different knowledge structure, such as graphs or lattices.

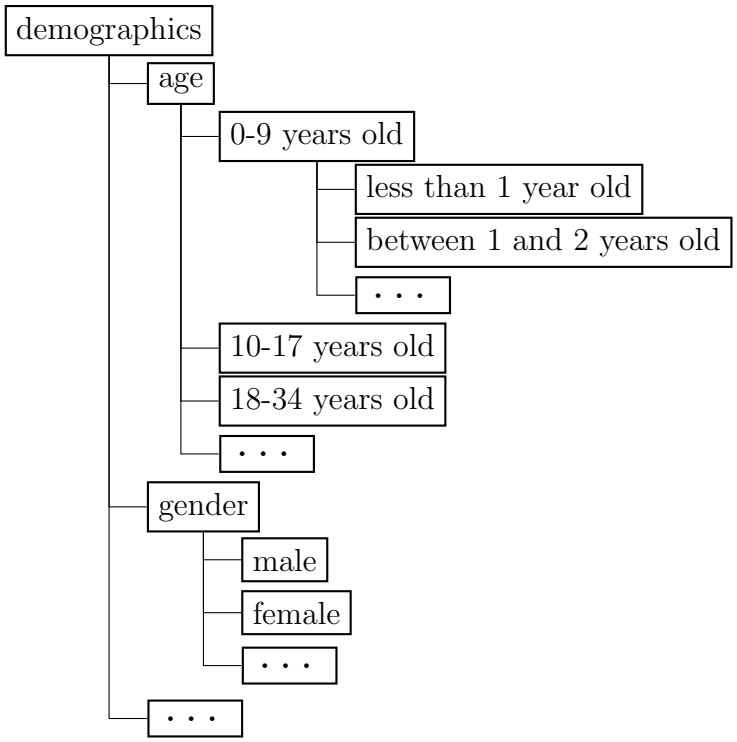


Figure 2.3: A small sample of facets available for patient medical records. For example, selecting female would query for all patients who are female.

Another example would be augmenting a faceted taxonomy with additional facets from an external faceted taxonomy. A more concrete example will help illustrate how such a situation can arise naturally. At our local university hospital, our data warehouse provides a faceted interface which allows individuals to obtain aggregate counts of patients who match facets selected by the user; this allows quick feasibility checks of clinical research projects. Facets are arranged as a simple hierarchy and include items such as demographics (age, race, marital status) and vital signs (height, weight, body mass index, blood pressure, heart rate, respiratory rate). These facets, illustrated in Fig. 2.3, are based on what the electronic medical records (EMRs) for our university hospital provides. In Fig. 2.3, lines represent relationships and ellipses imply there are some relationships not shown for space considerations.

The EMR also uses drug codes that link to an external proprietary system where drugs are organized as a two-level hierarchy. For example, as seen in Fig. 2.4, buprenorphine *is-an* analgesic and an analgesic *is-a* central nervous system agent. We include this external hierarchy as part of our faceted taxonomy by adding drug as a facet. This augmentation enables one to search for classes of drugs, rather than just the drug codes found directly in the patient's EMR.

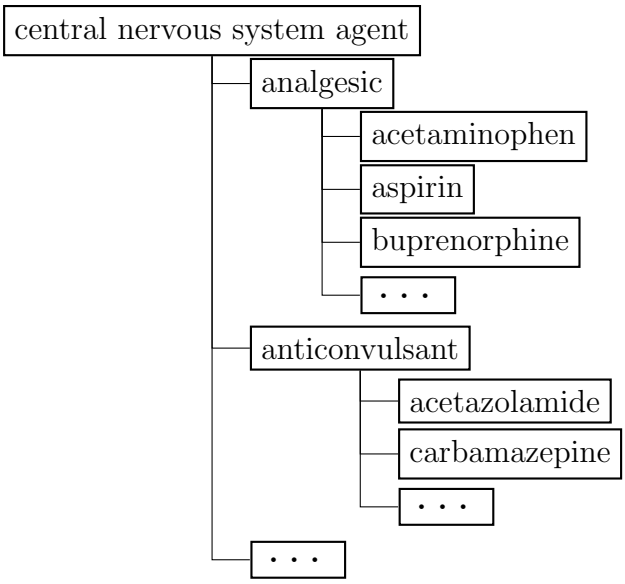


Figure 2.4: A small sample of facets available for drug administration records. For example, selecting analgesic would query for all records containing analgesics.

2.3 Category Theory

Category theory has been demonstrated to be practical and useful for modeling problems in the sciences [10], including physics [26], cognitive science [27], and computational biology [28]. In database theory, categories can model databases [29, 10] and can elegantly support data migration between schemas [30]. Additionally, ologs use category theory for representing knowledge and modeling real-world situations with the goal of enabling reusable, transferable, and comparable research [31]. Category theory can also be used in conjunction with semiotics as a foundation for information visualization [32], where the process of visualization forms a category.

Because our model leverages known categories, a working knowledge of category theory is necessary. Informally, a category $\mathcal{C}$ is defined by stating a few facts about the proposed category (specifying its objects, morphisms, identities, and compositions) and demonstrating that they obey identity and associativity laws.

Definition 1. *A category $\mathcal{C}$ consists of the following [10]:*

1. *A collection of objects, $Ob(\mathcal{C})$.*

2. *A collection of morphisms (also called arrows). For every pair $x, y \in Ob(\mathcal{C})$, there exists a set $Hom_{\mathcal{C}}(x, y)$ that contains morphisms from x to y [10]; a*

morphism $f \in Hom_{\mathcal{C}}(x, y)$ is of the form $f : x \to y$, where x is the domain and y is the codomain of f.

3. *For every object $x \in Ob(\mathcal{C})$, the identity morphism, $id_x \in Hom_{\mathcal{C}}(x, x)$, exists.*

4. *For $x, y, z \in Ob(\mathcal{C})$, the composition function is defined as follows: $\circ : Hom_{\mathcal{C}}(y, z) \times Hom_{\mathcal{C}}(x, y) \to Hom_{\mathcal{C}}(x, z)$.*

Given 1-4, the following laws hold:

1. *identity: for every $x, y \in Ob(\mathcal{C})$ and every morphism $f : x \to y$, $f \circ id_x = f$ and $id_y \circ f = f$.*

2. *associativity: if $w, x, y, z \in Ob(\mathcal{C})$ and $f : w \to x$, $g : x \to y$, $h : y \to z$, then $(h \circ g) \circ f = h \circ (g \circ f) \in Hom_{\mathcal{C}}(w, z)$.*

As an example, **Set** is the category whose objects are sets and whose morphisms are functions between sets [11, 10]. This implies that the set theoretic implementations of faceted browsing could also be directly abstracted and argued through category theory; we can comment on how our model could consume other models after we outline how our model works. The point of such structures is that they generalize sets by specifying families of elements rather than single elements, a formalization which enables exploration of structural similarities [10].

An easy way to construct a new category is to modify an existing category and create a subcategory by taking a subset of its objects and morphisms. One can think of this as simply removing some of the objects from the original category, then removing any morphisms that have lost their domain or codomain. The two categories behave similarly for those objects and morphisms found in both: they have the same identities and the same composition formula [11].

Definition 2. *A category $\mathcal{B}$ is a subcategory [11] of $\mathcal{C}$ if*

1. *$Ob(\mathcal{B}) \subseteq Ob(\mathcal{C})$*

2. *for all $x, y \in Ob(\mathcal{B})$, $Hom_{\mathcal{B}}(x, y) \subseteq Hom_{\mathcal{C}}(x, y)$*

3. *the identity morphisms and compositions for $\mathcal{B}$ are copied from $\mathcal{C}$.*

In our model, we will relate the concept of a facet and facet taxonomy to existing, well-known categories: **Rel** and **Cat**.

Definition 3. ***Rel*** *is the category of sets as objects and relations as morphisms [11], where we define relation arrows* $f : X \to Y \in Hom_{\boldsymbol{Rel}}(X, Y)$ *to be a subset of* $X \times Y$*.*

In other words, any subset of $X \times Y$ is a relation from X to Y. Any binary relation is allowed, but most examples demonstrate the utility of "$<$", "$\leq$", and "$\subseteq$". This category uses the composition of relations instead of functional composition; if $f \in Hom_{\mathbf{Rel}}(X, Y)$ and $g \in Hom_{\mathbf{Rel}}(Y, Z)$, then $(x, y) \in g \circ f$ if and only if for some $y \in Y$, $(x, y) \in f$ and $(y, z) \in g$. The identity morphisms, $id_X \in Hom_{\mathbf{Rel}}(X, X)$, are the so-called diagonal relationships $\{(x, x)|x \in X\}$. **Set** is actually a subcategory of **Rel** [11].

Definition 4. ***Cat*** *is the category of categories. The objects of* ***Cat*** *are categories and the morphisms are functors (mappings between categories).*

We informally defined functors as mappings between categories, but additional conditions are needed.

Definition 5. *A functor* F *from category* $\mathcal{C}_1$ *to* $\mathcal{C}_2$ *is denoted* $F : \mathcal{C}_1 \to \mathcal{C}_2$*, where* $F : Ob(\mathcal{C}_1) \to Ob(\mathcal{C}_2)$ *and for every* $x, y \in Ob(\mathcal{C}_1)$*,* $F : Hom_{\mathcal{C}_1}(x, y) \to Hom_{\mathcal{C}_2}(F(x), F(y))$*. Additionally, the following must be preserved:*

1. *identity: for any object* $x \in Ob(\mathcal{C}_1)$*,* $F(id_{\mathcal{C}_1}) = id_{F(\mathcal{C}_1)}$*.*
2. *composition: for any* $x, y, z \in Ob(\mathcal{C}_1)$ *with* $f : x \to y$ *and* $g : y \to z$*, then* $F(g \circ f) = F(g) \circ F(f)$*.*

Functors also play an important role in constructing the underlying graph of a category, which will be a key element of creating reusable facets and faceted structures.

2.3.1 Example Category and Usage

Recall that **Set** is the category whose objects are sets and whose morphisms are functions between sets [10]. Again, this implies that set theoretic implementations of faceted browsing could also be directly abstracted and argued through category theory. What can you do with **Set**? Categories often work as building blocks; in this example inspired by Spivak [10], list functions such as a concatenation are performed with **Set**. Specifically:

1. Let $X = \{a, b, c\}$, $Y = \{1, 2, 3\}$, and $f : X \to Y$ where $f(a) = 1$, $f(b) = 2$, and $f(c) = 3$. (a translation)

2. Let $List : \mathbf{Set} \to \mathbf{Set}$ be a functor that maps sets to lists of sets.

3. Let $List \circ List : \mathbf{Set} \to \mathbf{Set}$ be a functor that maps Lists to List of Lists.

4. Let μ_X be a natural transformation of Lists of Lists to Lists (concatenation).

The relationship and transformations possible between X and Y via $List$ operations are easily drawn. An example illustration is given in Figure 2.5 for $\Big[a, \big[b, c\big]\Big] \in List \circ List(X)$.

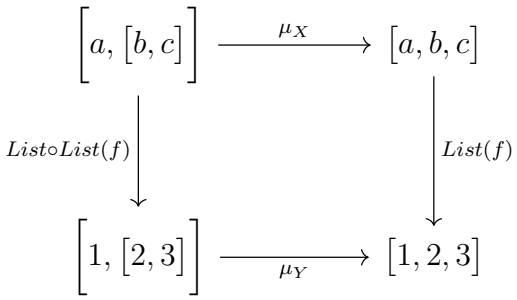


Figure 2.5: An example of **Set** and list manipulation.

It is important to note that order does not matter: the top right path concatenates then translates, while the bottom left path translates then concatenates.

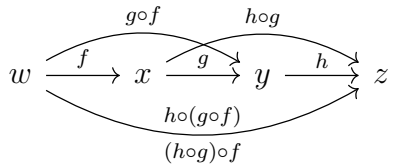
2.3.2 Why category theory?

We mentioned that category theory has been used to model several other practical problems in the sciences and will now discuss why it is also appropriate to model faceted browsing. The core issue is that faceted taxonomies come in many shapes and forms; this heterogeneity, while a sign that new and novel systems are being developed, is counterproductive for reusing faceted information effectively. Category theory provides the language for reasoning with these diverse structures in a consistent and productive environment. Once a category is defined, it becomes a bed for computations, such as transformations and products. As an example, consider n-ary products of objects within a category [11], which act as a categorical version of n-ary Cartesian products.

Definition 6. *The n-ary product of a list A_1, A_2, ..., A_n containing n objects (not necessarily distinct) of a category is an object A with morphisms $p_i : A \to A_i$ for $i = 1, \ldots, n$. This is denoted as $A_1 \times A_2 \times \ldots \times A_n$ or simply $\prod_{i=1}^{n} \boldsymbol{A}_i$ [11].*

We will demonstrate in the next chapter that n-ary products model faceted universes and faceted queries. Beyond products, category theory is conceptually consistent with faceted browsing in that relations within a faceted taxonomy naturally mimic the constraints of categories. Consider a faceted taxonomy with *is-a* relations and observe that the following analogies hold:

1. An object has an identity function: x *is-a* x.
2. Relations can be composed: (x *is-a* y) *is-a* z.
3. Commutative diagrams typically demonstrate how objects and morphisms in a category obey associativity:



   The same works for *is-a*-relations: ((*x is-a y*) *is-a z*) is equivalent to (*x is-a* (*y is-a z*)).

Chapter 3 A Category-theoretic Model of Faceted Browsing

A faceted system is comprised of many implicitly intertwined parts: facets, a taxonomy that organizes the facets, an ability to select or focus on certain facets, and an ability to present the results of a selection or faceted query in an effective manner [1]. Each of these components and their extensions can be abstractly modeled with category theory.

3.1 Taxonomies

We wish to be as general as possible in our abstractions so that any system with any faceted taxonomy can be modeled, regardless of the particular nuances of the facets and intra-facet relationships. There are two natural questions: how does one represent any taxonomy using category theory and how does one take that representation and augment with additional structure that supports the concepts of facets and faceted browsing? In Chapter 2, we introduced **Rel**, the category of relations, and we can restrict the morphisms of **Rel** to only correspond to the inclusion relations, which we henceforth refer to as $\subseteq$ relations:

Definition 7. ***Tax*** *is a sub-category of* ***Rel****, the category of sets as objects and relations as morphisms, such that* $Ob(\textbf{\textit{Tax}}) = Ob(\textbf{\textit{Rel}})$ *and morphisms are relations that correspond only to* $\subseteq$ *relations. The identity and composition definitions are inherited from* ***Rel****.*

The other relations that could possibly be represented by **Rel**, such as "$<$" and "$\leq$", are meaningless for categorically organizing concepts into a taxonomy. In other words, **Tax** is just a slimmer version of **Rel**, where we know exactly what binary relation is being used to order the objects. By itself, **Tax** is not particularly helpful for modeling faceted browsing: faceted browsing is piloted by interacting with disjoint collections of facets, which we will refer to as facet types. This will allow us to apply the additional structure and granularity needed to support faceted browsing.

3.2 Facet Types

The faceted taxonomy presented in an interface can contain several unrelated (or disjoint) sub-facets. For example, a book's price typically has nothing to do with

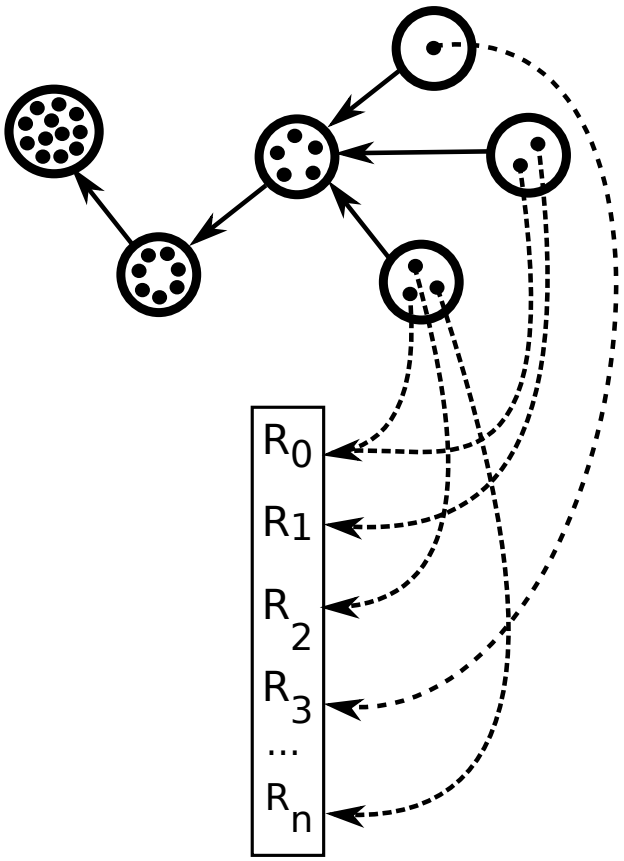


Figure 3.1: An illustrative example of a **Facet**$_i$ category shows a basic exemplative structure of the facets within this facet type. The objects are sets of pointers to resources classified as that particular facet type. For convenience, only pointers from leaf nodes have been drawn.

its genre, and the construction and maintenance of the corresponding taxonomic structures are completely independent. We can refer to a facet, such as price and genre, as a *facet type*. An interface typically presents facets underneath a heading that indicates its type. One may want the interface to include the name of the type as a selectable facet. This meta-facet is mostly an organizational tool that aids in drawing the faceted taxonomy. We formally defined meta-facets in Chapter 6 as part of our category-theoretic model of faceted browsing.

Within a facet type, facets are directly relatable and comparable. In other words, for our example, prices relate to prices and genres relate to genres; "\$10-\$20" can be a child of "$<$ \$100", but has no relationship with "horror". Figure 2.3 shows a facet type for patient demographics; Figure 2.4 shows a facet type for medications. Demographics and medications are clearly disjoint: no taxonomic relations exist between these two types.

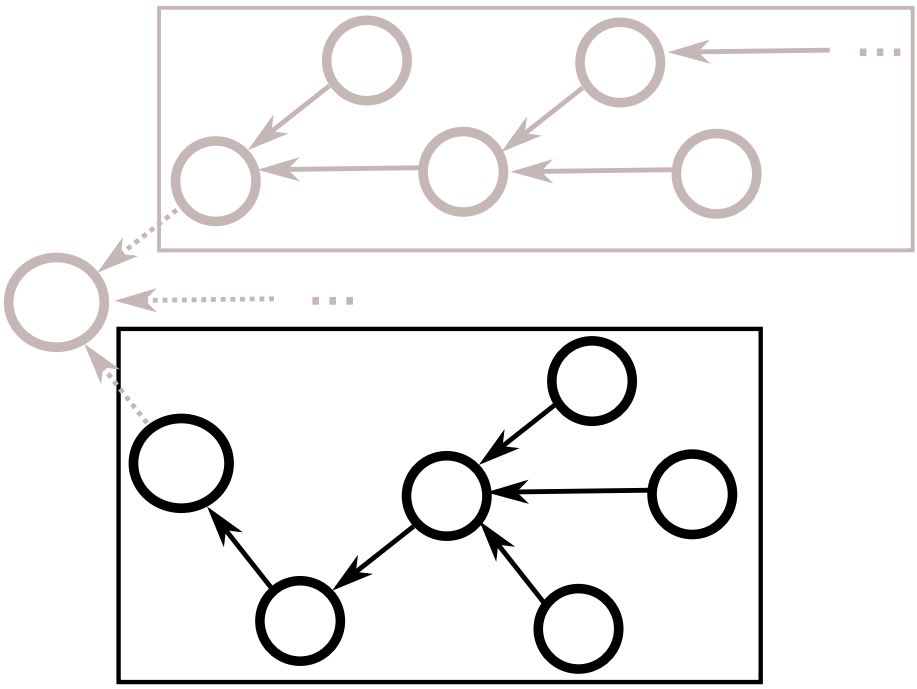


Figure 3.2: We draw the same exemplative facet type from Fig. 3.1 within the larger taxonomy. The facet types (illustrated as rectangular regions) act as groupings of objects that we can use as conceptual building blocks later in our model.

Theorem 3.1. *If there exists a category* ***Tax*** *having sets as objects and inclusion relations as morphisms, then there exists a subcategory of* ***Tax*** *called* $\boldsymbol{Facet}_i$*, representing facet types (a facet* i *and its related sub-facets).*

Proof. Take $\mathbf{Facet}_i$ to be a sub-category of **Tax** such that $Ob(\mathbf{Facet}_i) \subseteq Ob(\mathbf{Tax})$. For every pair of objects $(x, y) \in Hom_{\mathbf{Facet}_i}(x, y)$, there exists $(x, y) \in Hom_{\mathbf{Tax}}(x, y)$ that are inclusion mappings that relate a facet to its ancestor. Additionally, the relevant identity and composition definitions are inherited from **Tax**. □

As seen in Figure 3.1, the objects of $\mathbf{Facet}_i$ are sets which represent abstract collections of resources that have been classified to belong to that facet through faceted classification. With that being said, we do not need to distinguish between individual resources within these sets because they all categorically behave the same when manipulating objects: they either belong to a facet or they do not. Between the objects of $\mathbf{Facet}_i$, morphisms exist which dictate their taxonomic relationship. Figure 3.2 shows facet types in the context of a larger taxonomy; multiple facet categories exist and linkages between them may or may not exist.

As a concrete example, if $\mathbf{Facet}_{Med}$ is the medication facet-type displayed in Figure 2.4, then {"central nervous system agent", "analgesic", "buprenorphine", ...} $\in Ob(\mathbf{Facet}_{Med})$. The morphisms of $\mathbf{Facet}_{Med}$ dictate the relationships between objects. Suppose that object x is the set for "analgesic" and object y is the set for "central nervous system agent", then there exists a morphism $f : x \to y \in Hom_{\mathbf{Facet}_{Med}}(x, y)$, meaning "analgesic" *is-a* subset of "central nervous system agent".

The $\subseteq$ relation is powerful for specification: it allows for facets to be ordered by inclusion, which can model any structure where x is related to y; this is a pivotal component to most faceted implementations. We intentionally use the words specification and implementation; our category-theoretic model specifies the fundamental objects and relationships that our implementation can utilize. For example, we can implement a tree that only has *is-a* relationships since it is possible to order the facets by inclusion. We will show how this type of tree, which is heavily related to dynamic taxonomies [22], can be represented with category theory in Chapter 4. In a more complicated example, one could construct a visualization with a graph, where the current facet is recursively drawn connected to a subset of its subsumptive facets. In terms of designing an interactive interface for faceted browsing, the graph could be animated to highlight desired facets.

3.3 Focused Selections with Facets

Given a facet, we need to describe how any selection within the facet can be modeled.

Corollary 3.1.1. *We can create a subcategory of* $\boldsymbol{Facet}_i$*, called* $\boldsymbol{Focus}_i$*, to represent a focused selection of objects from* $\boldsymbol{Facet}_i$ *having* $Ob(\boldsymbol{Focus}_i) \subseteq Ob(\boldsymbol{Facet}_i)$ *and the necessary corresponding morphisms, identity, and composition definitions for those objects.*

We simply discard any undesirable objects (and their corresponding morphisms) to create a new category that represents a focused collection of facets. The identity and composition functions can be copied from $\mathbf{Facet}_i$. Selecting objects in facet type is the simplest form of interacting with a faceted browser. As a concrete example, if $\mathbf{Facet}_{Med}$ is the medication facet-type in Figure 2.4, then one possibility is {"central nervous system agent", "analgesic"} $\in Ob(\mathbf{Focus}_{Med})$. There is no limit on the number of objects kept or discarded; it is possible to discard all objects or to keep all objects, although practical limitations may stem from the user's ability to interact with the interface.

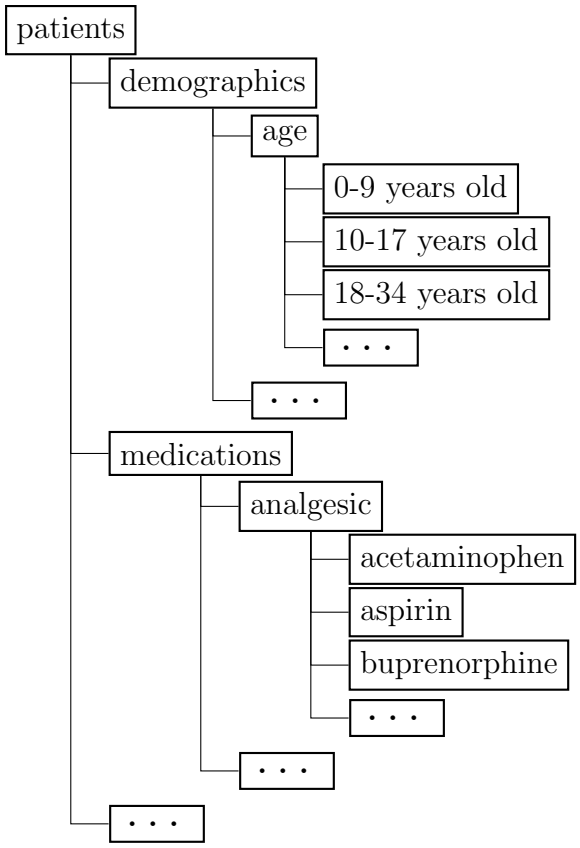


Figure 3.3: A faceted taxonomy, called patients, containing facet types of demographics and medications.

3.4 Faceted Taxonomies

Presentation of facets varies according to the interface's design, but facets are commonly presented as flat lists (through widgets such as radio buttons, check-boxes, etc) and hierarchies. Hierarchies, if restricted to be non-overlapping, are often represented as a tree where each facet is limited to having one parent. A non-hierarchical (or flat) facet is a special case of a hierarchical facet: it is simply a hierarchical taxonomy with one level. For example, if authors of books were a primitive facet, it could be represented as a hierarchy with author as the root and individual names as children. More complicated faceted interfaces may present taxonomies via visualizations, including graphs and network diagrams.

Theorem 3.2. *Suppose there exists categories for facet types* $\boldsymbol{Facet}_i, \boldsymbol{Facet}_{i+1}, \ldots, \boldsymbol{Facet}_n$*, then there exists a category* $\boldsymbol{FacetTax}$*, representing a faceted taxonomy containing* $\boldsymbol{Facet}_i, \boldsymbol{Facet}_{i+1}, \ldots, \boldsymbol{Facet}_n$ *as objects.*

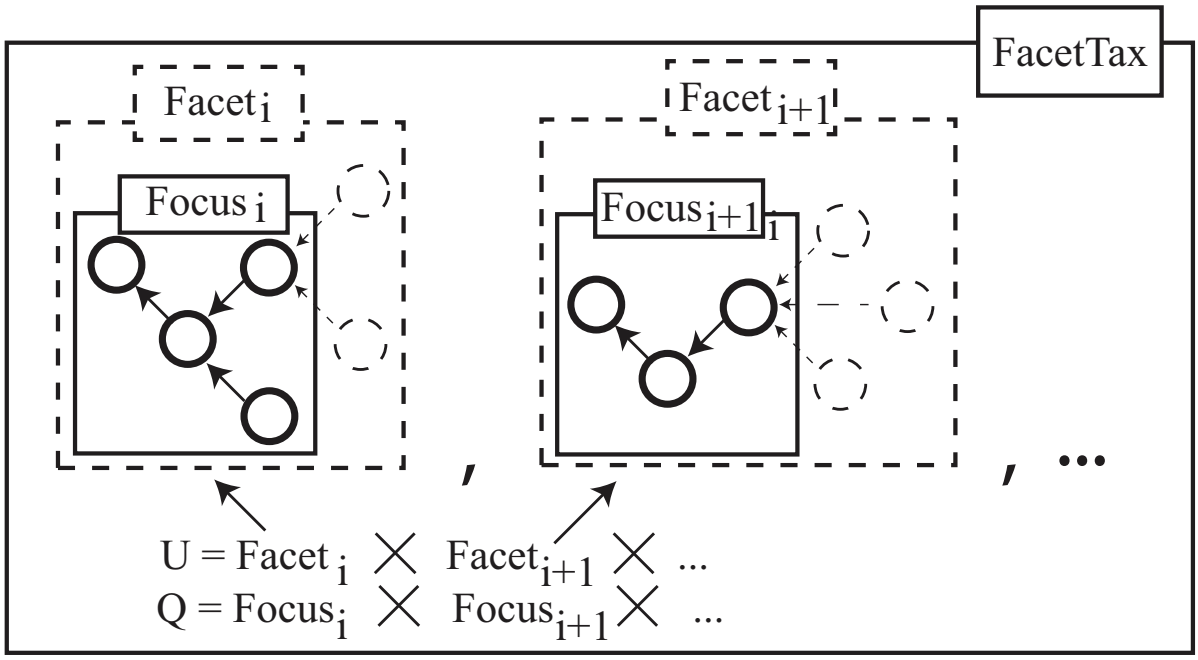


Figure 3.4: A high-level overview of **FacetTax**, **Facet**$_i$, **Focus**$_i$, and queries with focused subcategories.

Proof. Take **FacetTax** to be the disjoint union of **Facet**$_i$ categories, then the objects are defined as $Ob(\mathbf{FacetTax}) = \bigsqcup_{i=1}^{n} \mathbf{Facet}_i$ and $n = |Ob(\mathbf{FacetTax})|$. The morphisms of **FacetTax** are functors (mappings between categories) of the form $Hom_{\mathbf{FacetTax}}(\mathcal{C}, \mathcal{D}) = \{F : \mathcal{C} \to \mathcal{D}\}$. For any given $\mathcal{C} \in Ob(\mathbf{FacetTax})$, there exists an identity function because $\mathcal{C}$ is a category by definition. Given functors $\{F : \mathcal{C} \to \mathcal{D}\}$ and $\{G : \mathcal{D} \to \mathcal{E}\}$, then $G \circ F : \mathcal{C} \to \mathcal{E}$ by $Ob(G) \circ Ob(F) : Ob(\mathcal{C}) \to Ob(\mathcal{E})$ and $Hom_G \circ Hom_F : Hom_{\mathcal{C}} \to Hom_{\mathcal{E}}$. Furthermore, given functors $\{F : \mathcal{C} \to \mathcal{D}\}$, $\{G : \mathcal{D} \to \mathcal{E}\}$, $\{H : \mathcal{D} \to \mathcal{E}\}$. then $((H \circ G)) \circ F) = (H \circ (G \circ F))$. Given that identity is well-defined and composition is associative, **FacetTax** is a category. □

This disjoint union is precisely how we can merge facets from Figure 2.3 and Figure 2.4 into a single faceted taxonomy, *patients*, as illustrated in Figure 3.3. To save space, not all objects and relations are drawn. It is technically not necessary to formally prove that **FacetTax** is a true category because it is simply a sub-category of **Cat**, the category of categories. **FacetTax** acts as a highly structured version of **Tax**, where the objects are categories instead of objects; this additional structure allows us to model higher-level concepts such as faceted queries.

3.5 Facet Universe and Faceted Queries

The complexity of a faceted system naturally varies by the interface's design, but typically includes the ability to select (or focus) and de-select (or negate) facets within a facet type. The collective effort across all facets is then used to filter the faceted knowledge being presented.

Corollary 3.2.1. *A facet universe, U, is the n-ary product [11] within the* ***FacetTax*** *category, defined as $\prod_{i=1}^{n}$* ***Facet***$_i$*, where $n = |Ob($***FacetTax***$)|$. The n coordinates of U are projection functors $P_j : \prod$* ***Facet***$_i \rightarrow$ ***Facet***$_j$*, where $j = 1, \ldots, n$ is the jth projection of the n-ary product.*

Note that since $\mathbf{Focus}_i$ is a subcategory of $\mathbf{Facet}_i$, there exists a restricted universe $U_{\subseteq} \subseteq U$ where every facet is potentially reduced to a focused subset. The act of querying the universe is essentially constructing this restricted universe $U_{\subseteq}$.

Corollary 3.2.2. *A faceted query, Q, is the modified n-ary product [11] within the* ***FacetTax*** *category, defined as*
$\prod_{i=1}^{n}$ ***Focus***$_i$*, where $n = |Ob($***FacetTax***$)|$. The n coordinates of Q are similarly defined as projection functors $P_j : \prod$* ***Focus***$_i \rightarrow$ ***Focus***$_j$.

A high-level overview of the interactions between $\mathbf{Facet}_i$, $\mathbf{Focus}_i$, and queries with the **FacetTax** category is illustrated in Figure 3.4. To summarize, a faceted taxonomy is a category which contains **Facet** categories as objects; each **Facet** category contains objects and taxonomic relationships between the objects. Intuitively, focused selections are contained within their larger, unfocused facet categories. The universe of possible facets provided by the interface is the product across all **Facet** objects; a faceted query is simply a focused product chosen by the user from the universe of facets.

This implies that our model can represent states of interfaces that a user has interactively reached, which is a pivotal step in abstracting the interaction process within a faceted browsing system. Our model attempts to fill a void in human-computer interaction by providing abstractions that can consistently represent components of an interface for algebraic manipulation. Our model takes advantage of how intertwined a faceted browsing system is with its underlying faceted taxonomy and allows us to extend structural abstraction of a faceted taxonomy into the abstraction of faceted interfaces in general.

3.6 Pullback Operations

Recall that the objects of each $\mathbf{Facet}_i$ categories are sets of pointers toward resources which have been classified as belonging to a particular facet. Our model can create higher-level faceted groupings from existing facets by leveraging categorical pullback operations.

Corollary 3.2.3. *Categorical pullback operations, also known as fiber products [10], model interactive conjunctions of* $\boldsymbol{Facet}_i$ *and* ***FacetTax*** *categories, yielding new facet types that are not available directly in the taxonomy.*

In order to demonstrate creating new facet types from conjunctions, we must formally define pullbacks.

Definition 8. *Given sets* A, B, $C \in Ob(\mathcal{C})$ *for some category* $\mathcal{C}$*, a pullback of* A *and* B *over* C *is any set* D *where an isomorphism* $A \times_C B \to D$ *exists for* $A \times_C B = \{(a, b, c) | f(a) = c = g(b)\}$*; this is illustrated below, using Spivak's* $\lrcorner$*-notation to label the pullback [10]:*

$$(a) \quad \begin{array}{ccc} & & B \\ & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & C \end{array} \qquad (b) \quad \begin{array}{ccc} A \times_C B & \xrightarrow{\pi_2} & B \\ {\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & C \end{array} \tag{3.1}$$

The result of a pullback is easily illustrated with an example. If *horror* and *comedy* belong to the facet type for genres of either movies or books, then we can draw the relationships between *horror* and *comedy* easily:

$$\begin{array}{ccc} & & Horror \\ & & \downarrow{\scriptstyle is} \\ Comedy & \xrightarrow[is]{} & Genre \end{array} \tag{3.2}$$

We derive a new set that we can label *horror and comedy* by applying the pullback to the set of *horror* and the set of *comedy* objects:

$$\begin{array}{ccc} \text{Horror and Comedy} & \xrightarrow{\pi_2} & Horror \\ {\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle is} \\ Comedy & \xrightarrow[is]{} & Genre \end{array} \tag{3.3}$$

This forms a new set of objects being characterized by a conjunctive facet not directly found in the facet type; we could even give this new set a new semantic name: *comedic horror*.

If we apply the pullback to the set of *romance* and the set of *comedy* objects, we derive a new set that we can label *Romance and Comedy*:

$$\begin{array}{ccc} \text{Romance and Comedy} & \xrightarrow{\pi_2} & \textit{Romance} \\ \downarrow{\scriptstyle \pi_1} & \lrcorner & \downarrow{\scriptstyle is} \\ \textit{Comedy} & \xrightarrow[is]{} & \textit{Genre} \end{array} \tag{3.4}$$

Note that this forms a new set of objects being characterized by a conjunctive facet not directly found in the facet type; we could even give this new set a new semantic name: *romantic comedies* or *rom-coms*. This direct semantic name for groupings indirectly found within the data can become an engaging element of the interface and removes the possible limitation that only facets from the taxonomy can interact with the user.

The projection functions π_1 and π_2 may look trivial: a *comedic horror* title is clearly a comedy and clearly a horror title. Despite simplicity in appearance, the utility of the projection functions π_1 and π_2 mapping back to the original facets can be seen with faceted cues: for example, we can use π_1 to highlight *comedic horror* titles within the *horror* titles.

In Section 6.2.2, we introduce an existing faceted browsing system called i2b2 [14] as an example of an interface that requires multiple merged terminologies, but it is also an example of a faceted browsing system that targets healthcare data. A common goal within i2b2 is to identity groups of patient cohorts by dragging and dropping facets from a master taxonomy. A clinical researcher can quickly refine Boolean queries targeting patient populations; often these queries have a base population that can be specified as a conjunction. For example, a clinical researcher studying patients with breast cancer who have undergone a mastectomy procedure needs the ability to quickly reference such a population. We diagram what the data provides below:

$$\begin{array}{ccc} & & \textit{Procedure}(\textit{Mastectomy}) \\ & & \downarrow{\scriptstyle \text{belongs to}} \\ \textit{Diagnosis}(\textit{BreastCancer}) & \xrightarrow[\text{belongs to}]{} & \textit{Patient} \end{array} \tag{3.5}$$

If we apply the pullback to the category of *procedure (Mastectomy)* and the category of *diagnosis (Breast Cancer)*, we get a new category that we can label *Breast*

Cancer and Mastectomy:

$$\begin{array}{ccc} \text{Breast Cancer and Mastectomy} & \xrightarrow{\pi_2} & Procedure(Mastectomy) \\ \downarrow{\scriptstyle \pi_1} & \lrcorner & \downarrow{\scriptstyle \text{belongs to}} \\ Diagnosis(BreastCancer) & \xrightarrow[\text{belongs to}]{} & Patient \end{array} \tag{3.6}$$

This new category becomes an interactive element that can be reused within the interface; within i2b2, conjunctions can be annotated with a friendly human-readable name and can be shared across users.

Suppose we have a faceted taxonomy for patients that contains, among other facet types, patient procedures and diagnoses:

$$\begin{array}{ccc} & & Procedure(Amputation) \\ & & \downarrow{\scriptstyle \text{belongs to}} \\ Diagnosis(Diabetes) & \xrightarrow[\text{belongs to}]{} & Patient \end{array} \tag{3.7}$$

If we apply the pullback to the category of *procedure (amputation)* and the category of *diagnosis (diabetes)*, we get a new category that we can label *Diabetes and Amputation*:

$$\begin{array}{ccc} \text{Diabetes and Amputation} & \xrightarrow{\pi_2} & Procedure(Amputation) \\ \downarrow{\scriptstyle \pi_1} & \lrcorner & \downarrow{\scriptstyle \text{belongs to}} \\ Diagnosis(Diabetes) & \xrightarrow[\text{belongs to}]{} & Patient \end{array} \tag{3.8}$$

In fact, we can assign a new semantic name to this pullback, *diabetic amputees*, thus giving us facet types that do not appear directly in the faceted taxonomy. In the next section, we will demonstrate that pushouts help construct new facets from disjunctions.

3.7 Pushout Operations

Our model can assist in computing ad-hoc facets that attempt to compensate for short-comings in either the terminologies involved or the underlying data. For example, if the exact desired facet does not exist but its logical disjunctive components do, then it can be manufactured via pushout operations.

Corollary 3.2.4. *Categorical pushout operations, also known as fiber sums [10], model interactive disjunctions of* $\boldsymbol{Facet}_i$ *and* ***FacetTax*** *categories, yielding new facet types that are not available directly in the taxonomy.*

In order to show the expressiveness of pushout operations, we must formally define a pushout. We will then show how pushouts yield collections of objects projectable into **Focus** categories; in essence, pushouts within **Facet** demystify how **Focus** categories can be created interactively via selection.

Definition 9. *Given sets A, B, $C \in Ob(\mathcal{C})$ for some category $\mathcal{C}$, a pushout of sets B and C over A is any set D where an isomorphism $B \sqcup_A C \to D$ exists; this is illustrated below, using Spivak's $\ulcorner$-notation to label the pushout [10]:*

$$(a)\quad \begin{array}{ccc} A & \xrightarrow{g} & C \\ {\scriptstyle f}\downarrow & & \\ B & & \end{array} \qquad (b)\quad \begin{array}{ccc} A & \xrightarrow{g} & C \\ {\scriptstyle f}\downarrow & \ulcorner & \downarrow{\scriptstyle i_2} \\ B & \xrightarrow[i_1]{} & B \sqcup_A C \end{array} \tag{3.9}$$

It is important to note that $B \sqcup_A C$ was formed by the quotient of the disjoint union of A, B, C and an equivalence relation on B and C with A. An example will help demonstrate the utility of pushouts. If we return to our example with genre as a facet type, we can arrange our objects and morphisms so that the pushout reveals titles that are *comedy* or *romance*:

$$\begin{array}{ccc} Genre & \xrightarrow{has} & Comedy \\ {\scriptstyle has}\downarrow & \ulcorner & \downarrow{\scriptstyle i_2} \\ Romance & \xrightarrow[i_1]{} & \text{Comedy or Romance} \end{array} \tag{3.10}$$

This new facet, *comedy or romance*, contains all titles classified as either belonging to comedy or belonging to romance facets. The mappings i_1 and i_2 are sometimes called co-projections or inclusion mappings [10] and simply map the objects onto its larger disjunctive facet.

Pushout with FacetTax

Similar to our discussion on pullbacks with **FacetTax**, we can discuss pushouts. Again, the objects of **FacetTax** are complex **Facet** categories. If we revisit our faceted taxonomy example of a patient with diagnoses and procedures, the pushout becomes:

$$\begin{array}{ccc} Patient & \xrightarrow{has} & Diagnosis(Diabetes) \\ {\scriptstyle has}\downarrow & \ulcorner & \downarrow{\scriptstyle i_2} \\ Procedure(Amputation) & \xrightarrow[i_1]{} & \text{Diabetes or Amputation} \end{array} \tag{3.11}$$

We can call this new facet *diabetics or amputees*, again, giving a new semantic anchor to a class of resources that would not be directly found in the faceted taxonomy.

Hypertensive patients are woefully under-diagnosed and relying solely on diagnosis codes to locate patients with hypertension is problematic [33]. In addition to diagnosis codes, vital signs are either recorded by physicians and nurses or recorded by machines at given intervals; these measurements can be used to determine an individual's hypertensive state [33]. Recall that our resources for i2b2 are patients; patients can have diagnoses and vitals signs:

$$\begin{array}{ccc} Patient & \xrightarrow{has} & Diagnosis(Hypertension) \\ {\scriptstyle has}\downarrow & & \\ Vital(BP > 140/90) & & \end{array} \tag{3.12}$$

We compute the pushout and receive a single anchor for those individuals that were either coded to have a hypertension diagnosis code or that were recorded having high blood pressure:

$$\begin{array}{ccc} Patient & \xrightarrow{has} & Diagnosis(Hypertension) \\ {\scriptstyle has}\downarrow & & \downarrow{\scriptstyle i_2} \\ Vital(BP > 140/90) & \xrightarrow[i_1]{} & \text{Hypertension or BP} > 140/90 \end{array} \tag{3.13}$$

The pushout acts as a convenient, derived facet that the user can interact with just like any other facet. In i2b2, the disjunction between diagnoses codes and vital signs can be performed without the pushout because the interface itself allows for Boolean queries to be performed by dragging and dropping any facet into its query window. The value of the pushout is simply for convenience and reuse: when the pushout is used by multiple people, the context of a patient with hypertension is made clear and reusable.

We slightly modify the original diabetes example and substitute lab results for procedures:

$$\begin{array}{ccc} Patient & \xrightarrow{has} & Diagnosis(Diabetes) \\ {\scriptstyle has}\downarrow & & \\ Lab(HA1C > 6.5\%) & & \end{array} \tag{3.14}$$

In an electronic health system, not every diabetic patient will be assigned a diagnostic code indicating that they have diabetes; to compensate for this, clinical researchers often search for criteria that would imply that the patient has diabetes, i.e, patients

that either take medications known to treat diabetes or that have consistent lab results that indicate that they likely have diabetes [13]. For example, hemoglobin A1C test results above 6.5% are typically indicative of diabetes or early onset diabetes. If we construct a pushout operation, then we can help compensate for shortcomings in the data:

$$\begin{array}{ccc} Patient & \xrightarrow{has} & Diagnosis(Diabetes) \\ {\scriptstyle has}\downarrow & & \downarrow{\scriptstyle i_2} \\ Lab(HA1C > 6.5\%) & \xrightarrow[i_1]{} & \text{Diabetes or HA1C} > 6.5\% \end{array} \tag{3.15}$$

We can call this new set simply *diabetics*, as defined by those patients formally diagnosed with diabetes by a healthcare professional or those patients with laboratory results which indicate they likely have diabetes. The faceted taxonomy does not need to know ahead of time what possible disjunctions are relevant to the user; storing disjunctions would be ineffective when a lightweight pushout operation could derive the same result interactively with the user.

3.7.1 Focusing via Pushout Operations

Recall that **Focus** subcategories of **Facet** where $Ob(\mathbf{Focus}_i) \subseteq Ob(\mathbf{Facet}_i)$ are simply focused collections of objects. The disjunctive nature of pushouts make them an appropriate construction for realizing **Focus** categories from **Facet** categories.

Corollary 3.2.5. *Pushout operations create focused subcategories.*

Proof. Given a series of pushout constructions $p_0, \ldots, p_n$ having inclusions $i_{1p0}, i_{2p0}, \ldots$, we can take from the common domain of the projections to build the objects of **Focus**:

$$Ob(\mathbf{Focus}_i) = \bigsqcup_{j=1}^{n} dom(i_1 pj) \sqcup dom(i_2 pj)$$

The morphisms of **Focus** are simply those from **Facet** whose domain and codomain exist in **Focus** after construction.

□

3.8 Faceted Views

Returning to our previous healthcare example, there are existing standards for recording patient data, such as diagnoses and lab results. These standards can help form the faceted taxonomies behind clinical research interfaces designed to locate patients

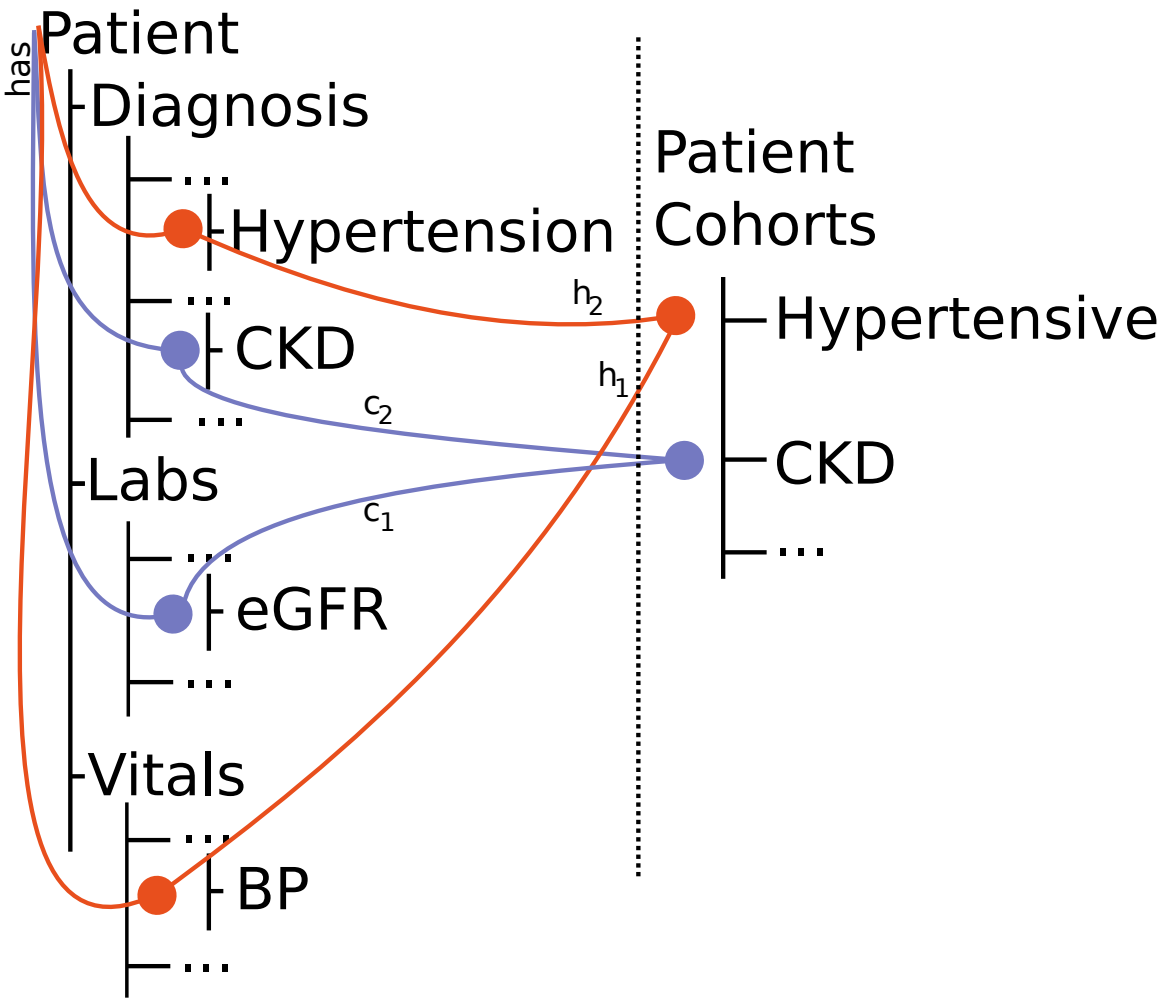


Figure 3.5: Faceted views can provide convenient, derived facets for interactivity.

of interest. Pushouts can derive faceted views on top of the taxonomy, providing desirable facets without disrupting or modifying the underlying standards. In other words, we can reuse data standards swiftly and construct an additional layer of facets that directly answers the needs of the interface; for example, this layer could be abstract representations of patients with diabetes, as determined by their medications and lab values as indicators of who likely has diabetes [13].

We can construct commonly-used patient cohorts via pullback and pushout operations: diabetic, hyptertensive, and chronic kidney disease (CKD) patients. CKD suffers from an issue similar to hypertension: diagnostic codes are not always used and might not capture the true disease state of the population, but laboratory results can predict CKD, including the disease's stage [34].

In Figure 3.5, we show that pushouts can create new facets from existing ones in

order to better address the needs of the interface. We create a facet for a hypertensive cohort by the pushout of diagnostic codes for hypertension and qualifying vital signs; we also create a cohort of CKD patients by considering CKD diagnostic codes and qualifying eGFR lab results. d_1 and d_2 are inclusion maps for the hypertensive cohort, while c_1 and c_2 are inclusion maps for the CKD cohort. The underlying taxonomies for patient data are large, having up to tens of thousands of nodes. The ability to create faceted views on top of the standard taxonomy will greatly improve the usability of the interface by providing the most meaningful facets to retrieve the desired resources.

Implementation of faceted views such as the one illustrated in Figure 3.5 requires the ability to represent multiple faceted taxonomies because patients can have diagnoses (from the ICD10 terminology) and vitals signs (from the LOINC terminology). We will expand upon this by introducing instances in Chapter 5.

3.9 Clarification of Benefits and Obstacles

In Section 2.3.2, we discussed why category theory is a good fit for modeling faceted structures in general; it is also important to identify what falls within the scope of our model and what does not. Our goal is to provide a common language and notation for describing faceted browsing interfaces in order for novel features or components to be reused intelligently and become interoperable with other features; we dedicate Chapter 4 to the notion of reusability within a system and we dedicate Chapter 5 to the notion of reusability across systems and implementations.

3.9.1 Efficiency and Scale

The objective of our model is not related to efficiency or speed of a particular interface, although we do discuss existing efforts in measuring computational complexity in Section 4.2.6. Our desire is to assist the developer of faceted systems in designing and developing in an efficient manner by encouraging reusable components, more-so than encouraging efficient components. The scale of the faceted taxonomy has no bearing on the abstractions offered by the model: a small taxonomy is a collection of objects with relationships just as much as a large taxonomy is a collection of objects with relationships. We have seen in our related work that the size of a very large taxonomy is still dwarfed by the size of the data that it describes [35]. To give an example of a large taxonomy, consider that the Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT) [36, 37] contains a few hundred thousand concepts describing

over a million medical terms; although large in comparison to other terminologies, the scale of SNOMED CT is often eclipsed by the size of the data it describes, which in the case of health data could contain billions of records [35].

3.9.2 Assumption of Classification Stage

Our model also assumes that the raw data has successfully undergone faceted classification so that assets are associated with facets. Our model is a theoretical facet model for faceted browsing and not faceted classification, which is a different process and falls into an alternative branch of faceted research. Of the existing systems identified and discussed in Chapter 2, none encompass faceted classification. The unstated assumption for these systems is that assets are classified and the system being discussed is exploring those assets in a novel and interesting manner. For biomedical data, facets such as demographics and disease states are either collected during a healthcare visit or assigned as a result of being billed for a healthcare visit. In this sense, the patient records are already classified according to how the electronic medical record has recorded the basic facts and observations of their medical visit.

3.9.3 Assumption of Taxonomy Existence

Because the data is already classified with facets, we also assume that a faceted taxonomy appropriate for the data set exists. The relationships in faceted browsing systems are largely taxonomic with parent-child relationships that provide structure to the resources being browsed, but any faceted relationship that can be represented as a graph can be utilized. In Chapter 4, we explore how graphs interact with our model. Facet extraction and automation of the construction process of a taxonomy is a different area of faceted research, as identified by Wei and as summarized in Table 2.1. These other areas of faceted research can be explored in addition to our theoretical model and would consequentially impact how algorithms are developed.

3.9.4 Barriers to Adoption

The largest barrier for adoption of our model is the willingness of others to use a model that requires a basic understanding of category theory. We acknowledge that the learning curve for category theory can certainly be an impediment of adoption by those less familiar with the theory; we argue that any solution that bridges barriers across models of faceted systems will require careful abstraction and will consume just as much energy to understand and to be consistent and correct. We believe

the benefits, such as the arguments of reuse and interoperability presented next in Chapter 4, outweigh the obstacles. The emphasis on reuse allows us to construct category-theoretic representations of existing facet models and systems; if we want to incorporate a novel component or feature of an existing facet model, we are able to view it through a category-theoretic lens. This implies that the barrier to adoption is not necessarily on those designing faceted systems but rather on those who wish to integrate, unify, and reuse multiple systems.

Chapter 4 Reusability

Our category-theoretic model enables reusability at two different levels: within a faceted browsing system and across faceted browsing systems. We will discuss each of these types of reuse in the following sections and describe why category-theory plays a fundamental role in each.

4.1 Reuse within a Faceted Browsing System

Because our model's category theory foundation, we may reuse objects within a faceted category and change the structure of their relationships according to the needs of the faceted system. For example, if it is more convenient to represent facets as graphs instead of sets in order to produce a visualization, we are free to compute the graph representation and category theory gives us the precise language to do so. In our model, categories are acting as generic faceted structures, but in practical interface development, more is sometimes needed to support the range of possible designs and interactions. The generic nature of the morphisms of **FacetTax** allow it to abstractly represent any faceted taxonomy structure since the morphisms are simply $\subseteq$ relations.

We demonstrated that n-ary products were one useful computation enabled by our model, but we can also demonstrate how our base structure can transform to support different faceted structures, such as lists, hierarchies, trees, graphs, and lattices. This transformation can be an active and engaging element of the interface: a selected element from a basic list could render a graph of deeper relationships. Using Figure 3.3 as an example, suppose that a preview of high level facet types are given in the form of a list (*demographics, medications, ...*) and interactively selecting one of these higher level items results in the rendering of its remaining taxonomy. For example, clicking on *demographics* could draw the descendants of that object from the taxonomy in the interface.

Intelligent previews are a possible solution to the problem of having too little screen space to fully display an interface's facets, which is a known issue in faceted browsing research [1]. We can reuse the same facets, while manipulating their relations to fit other structures. These structures can be put into the same abstract framework to support interoperability between systems or between parts within a system. When designing a system where different structures interact with one an-

other, the notation and representations become cumbersome. The burden of writing consistent abstractions with different structures is removed by using category theory.

4.1.1 Underlying Graphs

In order to show how graphs relate and interact with our model, we must formally define the category of graphs, for which we follow [10].

Definition 10. ***Grph*** *is the category with graphs as objects. A graph G is a sequence where $G := (V, A, src, tgt)$ with the following:*

1. *a set V of vertices of G*
2. *a set A of edges of G*
3. *a source function $src : A \to V$ that maps arrows to their source vertex*
4. *a target function $tgt : A \to V$ that maps arrows to their target vertex*

An alternate way to represent a category is by noting it as a sequence of its constituent parts, e.g, a category $\mathcal{C} = (Ob(\mathcal{C}), Hom_{\mathcal{C}}, dom, cod, ids, comp)$, where $dom, cod : Hom_{\mathcal{C}} \to Ob(\mathcal{C})$ are domain and codomain functions, *ids* are identity functions, and *comp* are compositions. This notation easily lets $\mathcal{C}$ be represented as a graph; in fact, some literature begins discussing categories as graphs first [11]; in this context, morphisms are typically called arrows and they map from a source (domain) to a target (codomain).

Definition 11. *The graph underlying a category $\mathcal{C}$ is defined as a sequence $U(\mathcal{C})$ = ($Ob(\mathcal{C})$, $Hom_{\mathcal{C}}$, dom, cod) [10].*

Note that $U(\mathcal{C}) \in Ob(\mathbf{Grph})$ and there exists a functor between categories $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ that can create a graph morphism $U(\mathcal{F}) : U(\mathcal{C}) \to U(\mathcal{D})$. This works at two levels for our model: for **FacetTax** and for each **Facet** category. Given that there exists a functor $U : \mathbf{Cat} \to \mathbf{Grph}$, **FacetTax** can produce graphs of $\mathbf{Facet}_i$ categories for $i = (1, \ldots, |Ob(\mathbf{FacetTax})|)$.

Because every category has an underlying graph, each individual **Facet** can also be represented as a directed graph, where the objects are vertices and the morphisms are arrows from one vertex to another vertex. Although the taxonomy in Figure 3.3 is likely best visualized as a simple hierarchy, other taxonomies might be more naturally suited to be represented as a graph. One possible use case is where graph algorithms might play a key role in the interface's design.

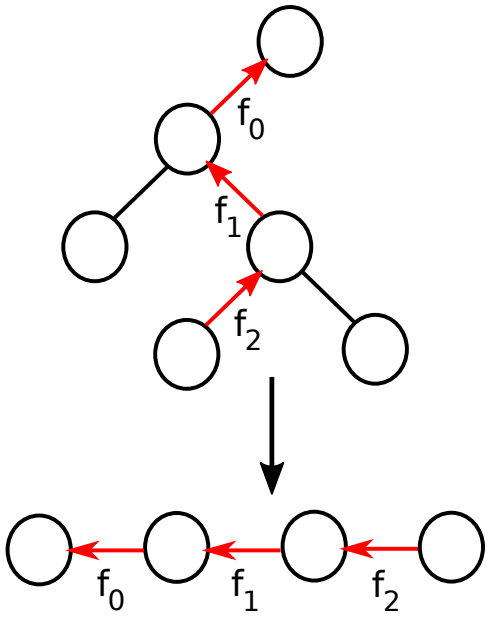


Figure 4.1: For graph-based faceted interfaces, paths are a simple way to abstractly model breadcrumbs. The above shows a breadcrumb f from a head-to-tail sequence of f_0, f_1, and f_2 arrows from a **Focus** category in **FacetTax**.

The key benefit to modeling our faceted taxonomy through category theory is that we can reuse the facets and re-frame their relationships (or morphisms) to fit our needs; if we need to arrange the facets as graphs, we can do so. The sets of resources that the objects of **Facet** represent remain unchanged; resources are still classified with their facets as illustrated in Figure 3.1. This embedded, faceted information can be reused through an alternate lens made possible by our model. In this case, the lens is a graph that was algebraically constructed, but other structures are also supported such as sets and lattices. Each structure has its own set of benefits; for example, graph representations allow the study of paths. The model remains agnostic to what representation is most desired; the important notion is that representation can change to adapt to the needs of the interface.

In the next section, we will discuss paths and comment on using sets and lattices. Using category theory as a common language ensures interoperability within a model even when using different faceted structures.

4.1.2 Paths

When considering the graph underlying **FacetTax** or $\mathbf{Facet}_i$, paths can play a role in modeling interactive components within an interface.

Definition 12. *If $G := (V, A, src, tgt)$ is a graph, then a path of length n in G, is a head-to-tail sequence of arrows and is denoted $p \in Path_G^{(n)}$, where $Path_G$ is the set of paths of any length in G [10].*

In Figure 4.1, we use paths as a simple example for modeling breadcrumbs, which have been identified as a key element of faceted search interfaces [38]. In this case, a path within a focused subcategory represents a possible navigation route from the set of all navigation paths, $Path_{\mathbf{Focus}}$. We will show in Section 4.2.4 that paths can be used in useful calculations within a category.

4.1.3 Sets

Rel is closely related to **Set**; both categories have sets as objects. In fact, it can be demonstrated that **Set** is a subcategory of **Rel** [11] and we will utilize this notion to show that each $\mathbf{Facet}_i$ is compatible with **Set**. We can construct a functor $F : \mathbf{Facet}_i \rightarrow \mathbf{Set}$, where $F : Ob(\mathbf{Facet}_i) \rightarrow Ob(\mathbf{Set})$ and for every $(x, y) \in Hom_{\mathbf{Facet}_i}(x, y)$ where x is related to y, we can construct a function $f(x) = y$ where an object is mapped to its ancestor in **Set**.

Similar to how we can construct graphs, we are also free to leverage **Set** categories and can construct basic sets. This is helpful when a flat list needs to be constructed from our taxonomy, such as in the case of the previews discussed in the first part of Chapter 4. In Section 4.2.1, we discuss several existing facet models that use set theory as a theoretical foundation and demonstrate how our categorical model can enable their reuse.

4.1.4 Other Structures

Additionally, **Rel** is capable of representing lattices when ordering by inclusions [11] and a similar result can be obtained with our **Facet** categories. Again, a functor is how we can compute such a mapping between categorical representations. Category theory is providing two types of computation: within a category with concepts, such as products, and across categories with concepts, such as functors.

4.2 Reuse across Faceted Browsing Systems

We demonstrated that facets can be reused within a single system in the previous section because category theory gives us the ability to rearrange the structure of the relationships as needed. In this section, we focus on reusing components and concepts

across faceted browsing systems. The essential requirement for reusing components is that a common language exists that can bridge across components and enable communication and interactivity between the two.

4.2.1 Reusing Existing Facet Models

We discussed several existing modeling efforts in our background section and we can now discuss how such models would change and manifest under our category-theoretic model.

4.2.2 Dynamic Taxonomies

Dynamic taxonomies [22] use set theory as a theoretical foundation to help model facets. The model constructs taxonomies with *is-a* relationships by dynamically calculating the deep extension of a node:

$$\text{deep}(C) = \{d | d \in \text{shallow}(C\prime) \\ \wedge (C\prime = C \vee C\prime \text{ is a descendant of } C)\} \quad (4.1)$$

The shallow extension of C contains the direct descendants of C. Both shallow extension and *descendant-of* relationships are expressible as binary relations and, in particular, are expressible with the binary relations of **FacetTax**'s **Facet** categories. The shallow extension of an object $y \in Ob(\mathbf{Facet}_i)$ is the domain of the relations of $Hom_{\mathbf{Facet}_i}(x, y)$, e.g. all x that are subsets of y:

$$\text{shallow}(y) = \bigcup_{\substack{x \in Ob(\mathbf{Facet}_i) \\ x \neq y}} dom(Hom_{\mathbf{Facet}_i}(x, y)) \quad (4.2)$$

Intuitively, we can think of the deep extension as the nested shallow extension, meaning we are free to construct $Hom_{\mathbf{Facet}_i}(x, dom(Hom_{\mathbf{Facet}_i}(y, z)))$ and so on. We can formalize this as a recursive union, so for any $y \in Ob(\mathbf{Facet}_i)$, we must look at all $x \in Ob(\mathbf{Facet}_i)$ where x is in the shallow extension of y.

In other words, to calculate the deep extension of y, we must recursively aggregate all x where x is a direct descendant of y:

$$\text{deep}(y) = \bigcup_{\substack{x \in Ob(\mathbf{Facet}_i) \\ x \in \text{shallow}(y)}} \text{shallow}(y) \ \cup \ \text{deep}(x) \quad (4.3)$$

The recursion stops when a leaf node is touched and the shallow extension of the object is the empty set. The leaf nodes only have an identity morphism where the

domain and codomain is itself; this morphism is sometimes omitted when drawing taxonomic relations. We exclude this morphism by constraining the shallow extension to consider only those $x \in Ob(\mathbf{Facet}_i)$ where $x \neq y$, which in turn ensures that the recursion will end when calculating the deep extension. Ultimately for a leaf node y, there will be no $x \in Ob(\mathbf{Facet}_i)$ such that x is in the shallow extension of y. We are free to also include or exclude an object in the deep extension of itself, depending on what is most convenient for the faceted interface's design.

4.2.3 Category Hierarchies

Category hierarchies are another example of facet models with set theory as a foundation [23]. In this model, category hierarchies are defined as connected and rooted directed acyclic graphs; the word category is unrelated to category theory in this context.

More specifically, a category hierarchy is defined as a sequence $H(r_H, C_H, E_H)$ representing a rooted and directed acyclic graph, where r_H is a designated root category, $C_H = \{c\}$ is the set of categories in the hierarchy, and $E_H = \{c \rightarrow c\prime\}$ is the set of category to subcategory relationships [23]. Facets are defined as subgraphs of the category hierarchy. We add a root category to our definition for underlying graphs in Section 4.1.1 and build a new sequence where $H(Ob(\mathbf{FacetTax}), Hom_{\mathbf{FacetTax}}, dom, cod, r)$ represents the category hierarchy. Facets are objects of **FacetTax** instead of direct subgraphs, but we can reuse the underlying graph notation to define a facet as a sequence $F_i(Ob(\mathbf{Facet}_i), Hom_{\mathbf{Facet}_i}, dom, cod, r_i)$ to create the graph underlying the $\mathbf{Facet}_i$ category at some root r_i. At this point, we are free to leverage the deeper parts of the model using our facet graphs, such as measuring cost of navigational paths and facet similarity.

4.2.4 FaSet

There exist large differences between facet models even if they share the same theoretical foundation such as set theory. In FaSet [25], a facet F is a set of items; in systems with multiple facets, they are disjoint: $F_a \cap F_b = \emptyset$. The items in the set represent labels for that facet. A focus L is a named subset of $F : L \subseteq F$, where the name is a nullable, variable-length list of indexes: $L\langle i, j, k, \ldots\rangle$. The classification of a resource r is the subset of F that is relevant, denoted as $r \perp F$. A sharp classification is a classification for some set of focus names P that can be expressed as a union of

foci: $\exists P : r \perp F = \bigcup_{p \in P} L\langle p \rangle$. This model allows classifications to be written as easily implementable lists.

FaSet is purely a set-based model of faceted browsing. Differences between the models must be reconciled in order to inject the features of FaSet into **FacetTax**. The following differences must be noted:

1. The most primitive difference is that in FaSet a facet is a set of items (labels) while in **FacetTax**, a facet is a category. Many of the following differences are a direct result of **FacetTax** containing objects and relationships; mainly, it does not require external operators and special notations to reference relationships.

2. Because of FaSet's set-based foundation, its model must provide a set-based method of classifying resources with facets; it introduces custom notation to achieve this: $r \perp F$ for classifying a resource r with facet F. The objects of **FacetTax** are sets and represent abstract collections of resources that have been classified as belonging to that facet through faceted classification. In other words, **FacetTax** embeds the link to resources into its structure; the relationships between objects dictate the nature of their taxonomic relationships.

3. In FaSet, focusing is performed by creating lists of indexes on a facet. They introduce an indexing notation $L\langle i, j, k, \ldots \rangle$ for a focus of L where $L \subseteq F$. We abstract the concept of focusing and create a **Focus** category.

4. Because sets are flat structures, FaSet must and successfully does demonstrate that it is capable of handling facets with varying structure (flat, hierarchical, nested) and dimension (single or multiple). This is not necessary with our model: any object can hold any relationship with any other object. In a category-theoretic model, we do not need to name specific objects and local relationships are generic between each object: x is-a y, which can ultimately create flat, hierarchical, and nested structures.

The primitive aspects of FaSet are replaced by the structure naturally given by **FacetTax** and the related $\mathbf{Facet}_i$ categories. This helps avoid the need for creating custom notation. Once conceptual differences are reconciled, key elements and extensions of the model can be rephrased. For example, calculating focus similarity of two foci of a facet depends on finding the depth of a focus, which is the number of hierarchy levels that compose the name of the focus. There are many ways to calculate focus depth when using our model, but a natural way is to represent this

depth as the largest path in the graph underlying $\mathbf{Focus}_i$, where paths are defined as in Section 4.1.1:

$$\text{depth}(\mathbf{Focus}_i) = \max_{p \in Path_{U(\mathbf{Focus}_i)}} (|p|) \tag{4.4}$$

4.2.5 Reuse and Unification of Efforts

Reusing existing models, or even parts of existing models, is a difficult endeavor due to the diversity of key concepts, definitions, and notations. Our goal in discussing how our model relates to existing work is to demonstrate how dynamic our category theory foundation can be and that it is able to drive interoperability and reuse. Resulting work can provide for the reuse and merging of existing models by uniting them in a common language. In this case, theory has direct consequences for specification and can help drive implementation.

4.2.6 Computational Complexity

We have focused on structural complexity between categories and morphisms, but computational complexity should also be addressed. It is only appropriate to consider computational complexity in the context of using abstract categories to write algorithms, which is one way our model can be applied and reapplied. An example is computing the transitive closure of **Fin**, which can be solved as the repeated accumulation of pairs of paths and can be demonstrated as having a complexity of $O(N^3 \log N)$ where N is the number of nodes [39].

Computational category theory [39] connects functional programming with category theory to bridge the gap between theory and implementation. A by-product of computational category theory is that computational complexity can be studied.

4.2.7 Preliminary Notes on Implementation

Even elegant abstractions can be rendered useless if they are not easily implementable. Category theory is strongly related to functional programming; one can even show that a functional programming language forms a category of types and operations [11]. Furthermore, the use of objects (and morphisms between objects) as our model's foundation means implementation could be in any programming language that supports an object-oriented paradigm. For instance, Scala [40] is a multi-paradigm language which supports both functional and object-oriented paradigms. In fact, the category of finite sets, often called **Fin** [11, 10], is easily implementable in Scala [41]; there are Scala libraries available that provide infrastructure for building categories [42, 43].

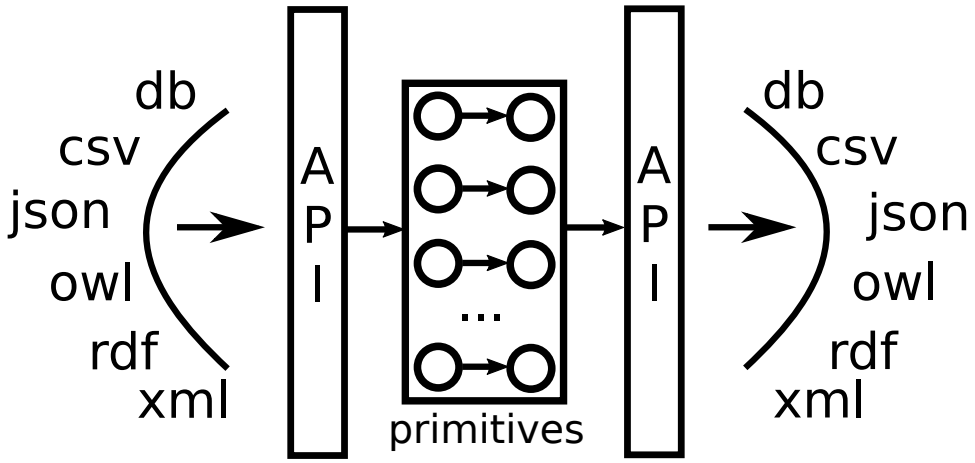


Figure 4.2: An API can wrap our model of facets where incoming faceted data is broken into primitives for abstract reasoning and logic; we can then export in a convenient format for interface development.

Our interests are in data-driven interfaces, so conceptually a database is a desired and necessary element. In the next chapter, we focus on showing that facets can be implemented using relational databases by mapping facets to schemas.

We contend that modeling has direct consequences for implementation; an inconsistent model yields an inconsistent system. If we cannot abstractly represent faceted browsing in an effective manner, it is difficult to extend and improve such a system and furthermore, the system cannot be adopted or easily modified by others. Theory must inform practice; we can use our abstractions to build stronger interfaces that support interoperability and reuse. Our model, together with category theory, can help inform how to build a proper application programming interface (API) for faceted browsing. In this vein, one can mathematically prove that something is possible before implementation.

As illustrated in Figure 4.2, an API for faceted browsing can provide methods for breaking down relationships from external file formats into primitive relationships (e.g., x *is-a* y) for abstract reasoning and logic. In the case of DELVE [8], faceted data from XML files are deconstructed into basic relationships; these relationships are then exported as a list in a JSON file that is compatible with a popular visualization library.

Our model does not intend to compete with existing and well-known standards such as the web ontology language (OWL) [44]. OWL in itself is not a facet model, but could be combined with abstractions to make it an appropriate solution. We later discuss the role of databases in facet browsing, but we could easily discuss OWL

as well. Because we are interested in database-driven web applications, we focus on databases instead. With respect to OWL, our goal is not to compete with as a standard, but rather to easily marry it with other types and deconstruct its relationships into usable primitives for the development of faceted browsing interfaces. We aim to leverage faceted data in any format into a consistent framework for abstraction.

Chapter 5 From Reusable Abstractions to Reusable Implementations

Even great abstractions are rendered useless for practical applications if they are not readily implementable. In this section, we detail how our model directly drives implementation; the result is that we can design faceted browsing systems with reusable abstractions and as a result produce reusable and interoperable implementations. We show that our category-theoretic facet model can be mapped to a known category representing database schemas, originally introduced by Spivak [10].

Given that there exists a mapping between facet and schemas, we give details on how facets can be implemented within a relational database. Implementation requires instantiation of our proposed abstractions, so we begin by introducing the concepts necessary to understand instances in the context of category theory.

An additional benefit of instances is that they allow us to model interfaces that leverage multiple taxonomies. The category-theoretic model is perfectly capable of representing basic faceted interfaces in its most basic form, but the ability to model and interact with multiple heterogeneous sources is needed to support more intricate interfaces. The capacity to integrate multiple terminologies rests largely upon our ability to model instances of our facet categories. Understanding the relationship between schemas and facets will be key in understanding the process for creating instances.

5.1 Facets and Schemas

In this section, we describe how to create instances of facets and faceted taxonomies with a method and rationale that is inspired by Spivak's database schemas [10]. In fact, we discover that facets are equivalent to database schemas because mappings exist between the two. Although this equivalence may be unexpected initially, conceptually the idea of a database schema is not unlike facets when viewed from a category theory perspective: both describe the conceptual layout that organizes information (rows/entities in the case of databases and resources in the case of facets).

Theorem 5.1. *If there exists an instance of a facet type* $\boldsymbol{Facet}_i$ *category, then there exists a corresponding instance of a database schema, representing this instance of a facet type.*

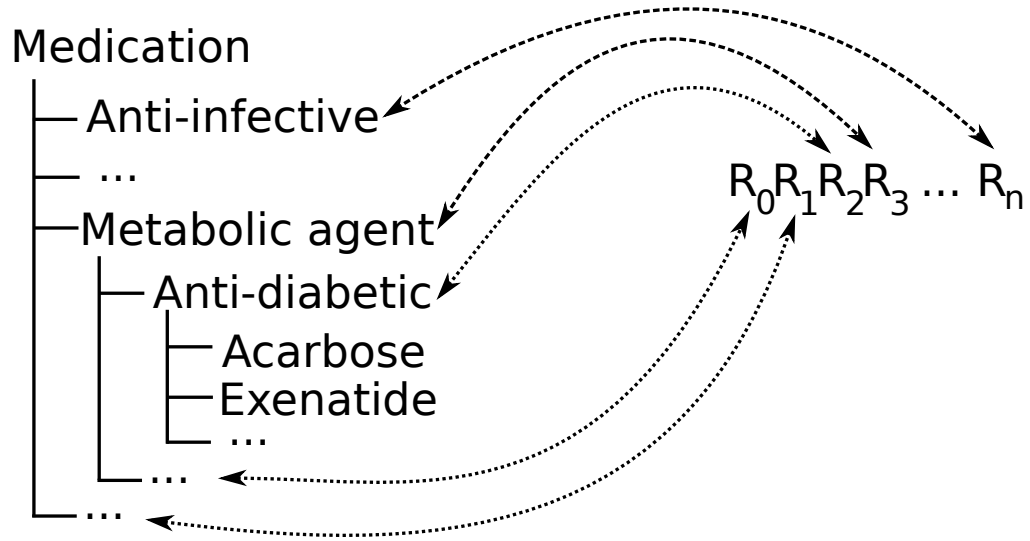


Figure 5.1: We show a sample faceted taxonomy for medications. The objects of each **Facet** are pointers to a resource that has been classified as belonging to that particular facet type.

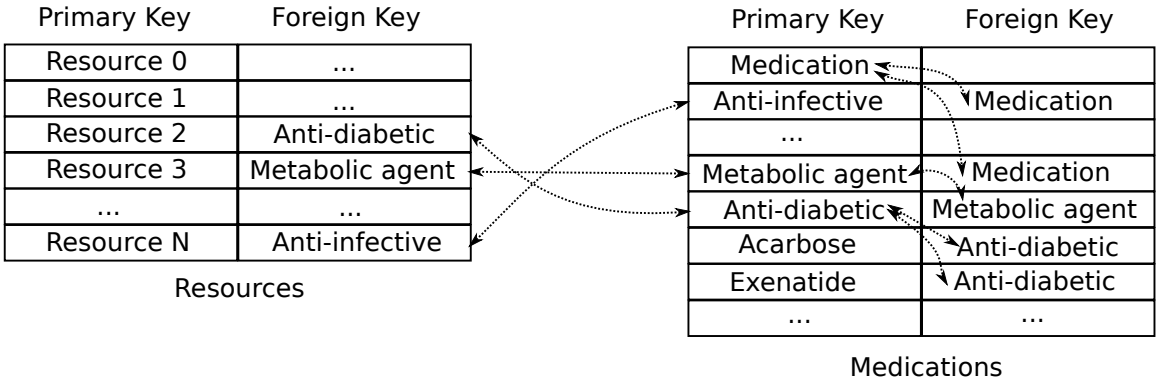


Figure 5.2: A resource table and a medications table using example data from Figure 5.1 shows the role that primary and foreign keys play in modeling faceted browsing.

In order to demonstrate this relationship, some preliminary definitions are needed and are discussed in the next section. As an example, Figure 5.2 shows the same faceted information found in Figure 5.1, but within a schema. Note that parts of the table are abbreviated with ellipses in order to save space. We will discuss these tables and their relationship with faceted browsing in detail in the next section.

Preliminary Definitions

Spivak's definition of schemas depends upon the idea of congruence, which in turn depends on defining paths, path concatenation, and path equivalence declarations [10]. We give the minimum required amount of detail to understand schemas below and extend upon this in our discussion of instances of **Facet**.

Definition 13. *If $G := (V, A, src, tgt)$ is a graph, then a path of length n in G is a sequence of arrows denoted $p \in Path_G^{(n)}$, where $Path_G$ is the set of paths in G [10].*

Definition 14. *Given a path $p : v \to w$ and $q : q \to x$, $p + +q : v \to x$ is the concatenation of the two paths [10].*

Definition 15. *A path equivalence declaration (abbreviated by Spivak as PED) is an expression of the form $p \simeq q$, where $p, q \in Path_G$ have the same source and target, e.g., $src(p) = src(q)$ and $tgt(p) = tgt(q)$ [10].*

Definition 16. *A congruence on G is a relation $\simeq$ on $Path_G$ with the following [10]:*

1. *The relation $\simeq$ is an equivalence relation.*
2. *If $p \simeq q$, then $src(p) = src(q)$ and $tgt(p) = tgt(q)$.*
3. *If given paths $p, p\prime : a \to b$ and $q, q\prime : b \to c$, and if $p \simeq p\prime$ and $q \simeq q\prime$, then $(p + +q) \simeq (p\prime + +q\prime)$.*

Informally, a congruence is an enhanced equivalence relation that marks how different paths in G relate to one another by enforcing additional constraints; pairing a graph with a congruence forms a schema [10].

Categorical View of Schemas

We give Spivak's definition of a schema below; this definition is generic enough to also apply to faceted browsing when looking at the underlying graph of the facet categories. Figure 5.2 contains a schema corresponding to the medications example from Figure 5.1.

Definition 17. *A schema S is a named pair $S = (G, \simeq)$, where G is a graph and $\simeq$ is a congruence on G [10].*

Note that the keys in Figure 5.2 would normally be integer keys, but here text labels are applied to increase readability and to improve the ease of understanding the example. The resource table in this schema contains a generic list of resources (for example, documents or library items) where each resource has a foreign key indicating how it is classified. The medications table contains a list of classes and sub-classes for medications, as well as a self-referential foreign key pointing back at itself; this foreign key indicates this particular medication's ancestor. The self-referential key gives additional structure to the medication classes and sub-classes found within the table without the need for additional relationship tables; this method of storing a taxonomy is similar to closure tables [45].

In Figure 5.2, the entry with *Medication* as its key has no foreign key. This null relationship indicates that this entry is the root of this particular facet graph; with respect to the category-theoretic model, it implies there are no morphisms having this object in its domain. This entry, formally introduced in the next chapter, is a meta-facet and is sometimes manufactured simply for convenience in arranging facets.

5.1.1 Instances of Facets and Faceted Taxonomies

An instance of a facet is a collection of objects whose data are classified according to specific relationships, such as the one illustrated in Figure 5.1. We formalize this below using Spivak's instances of schemas as inspiration [10].

Definition 18. *Let $F = (U(\boldsymbol{Facet}_i), \simeq)$, where the graph underlying a facet type is denoted $U(\boldsymbol{Facet}_i)$ for some $\boldsymbol{Facet}_i \in Ob(\boldsymbol{FacetTax})$ and where $\simeq$ is a congruence on $U(\boldsymbol{Facet}_i)$. An instance on F is defined as $(Facet, Ancestor) : F \to \boldsymbol{Set}$ where:*

1. *Facet is a function defined as $Facet : V \to \boldsymbol{Set}$, so for each vertex $v \in V$ we can recover a set of facets denoted $Facet(v)$ within this facet type.*

2. *for every arrow $a \in A$ having $v = src(a)$ and $w = tgt(a)$, a function $Ancestor(a) : Facet(v) \to Facet(w)$.*

3. *congruence is preserved: for any $v, v\prime \in V$ and paths p,$p\prime$ from v to $v\prime$ where $p = v[f_0, f_1, f_2, \ldots, f_m]$ and $p\prime = [f_0\prime, f_1\prime, f_2\prime, \ldots, f_n\prime]$, if $p \simeq p\prime$, for all $x \in Facet(v)$, $ancestor(f_m) \circ \ldots \circ ancestor(f_1) \circ ancestor(f_0)(x) = ancestor(f_n\prime) \circ \ldots \circ ancestor(f_1\prime) \circ ancestor(f_0\prime)(x) \in Facet(v\prime)$*

Corollary 5.1.1. *Primary keys can store facets and self-referential foreign keys can store ancestor relationships.*

Remark. *Given $F = (U(\boldsymbol{Facet}_i), \simeq)$, an instance on F, denoted as a pair of functions $(Facet, Ancestor)$ is mappable to an instance of schema, denoted as a pair of functions $(Primary\text{-}Key, Foreign\text{-}Key)$, by forcing the $Primary\text{-}Key$ function to represent the corresponding $Facet$ and the $Foreign\text{-}Key$ function to represent the corresponding $Ancestor$.*

It is important to note that because identity functions are enforced by definition of a category, one record in the database must store a link between a facet and itself. This type of a data structure is useful when calculating closure tables within a relational database [35].

Instances of categories are often denoted as I where $I_0, I_1, \ldots, I_N$ would be a collection of N instances.

To create instances of **FacetTax**, the logic remains the same from **Facet**: take the underlying graph and a congruence. Instead of looking at the underlying graph of a single facet type, the underlying graph of **FacetTax** is considered.

5.2 Implementing Facets with Databases

If we connect instances back to our notion that schemas are not structurally different than facets, it is clear that I_M is simply another table containing $N + 1$ relationships with entries from the $\mathbf{Facet}_0, \ldots, \mathbf{Facet}_N$ categories sharing a relationship with the meta-facet. The foreign keys of these meta-relationships would simply point back to the roots of the other facets; this enables reuse in-place without needlessly copying data.

Furthermore, this gives a clear implementation path for enabling reusable terminologies in a standard relational database, where tables help structure facets and the resources that have been classified accordingly. If a relational database is not possible for the application, then an equivalent scheme can be mimicked in other suitable environments that can store hierarchical data. For example, a web-application could easily use JSON (Javascript Object Notation) data interchange format [46] to store the taxonomy and links to resources.

5.2.1 Database Implementation

In this section, we give a concrete interpretation of the mapping illustration in Figure 5.2 and assume that we are storing our faceted data within a relational database.

It is important to note that at a minimal the model requires an identifier for itself and a self-referential foreign key pointing to itself to represent the ancestor-descendant relationships. We are free to add additional columns if they are convenient, such as a human-readable label as seen in the following definition of an example facet:

```
CREATE TABLE facetX
(
    id INTEGER,
    label TEXT,
    ancestor INTEGER references facetX(id),
    constraint fx_pkey primary key (id)
);
```

The label is optional but will simplify interactivity later. In addition to the table storing the faceted structure, there needs to be a table for storing relationships between the resources and the facets. Every resource at a minimum needs an identifier and when coupled with a facet's identifier, we get distinct pairs of resource-facet relationships:

```
CREATE TABLE resources
(
        resource_id INTEGER references resources(id),
    facet_id INTEGER references facetX(id),
    constraint resrc_pkey primary key (resource_id, facet_id)
);
```

We generalize these statements to support multiple faceted taxonomies in the next chapter. We use these generalizations to assist in the implementation of DELVE's components in Chapter 7.

5.2.2 Recalling Resources

At some point during a user's interactive session in a faceted browsing system, it is advantageous or desirable to recall and list all resources that were classified according to a focused selection of facets. When creating instances of our facet categories, we defined a function capable of returning the ancestor of the facet type for a given facet. We can similarly define a function capable of returning focused resources.

Definition 19. *Let R be a function defined as $R(Focus, Resource) : \textbf{\textit{Focus}} \rightarrow \textbf{\textit{Set}}$, where:*

1. *$Focus$ is a function similar to the $Facet$ defined in Section 5.1.1: $Focus : V \rightarrow \textbf{\textit{Set}}$, so for each vertex $v \in V$ we can recover a set of focused facets denoted $Focus(v)$*

2. *$Resource$ is a function defined for every focused facet $f \in Focus(v)$ above as $Resource(f) : Focus(v) \rightarrow Resource(f)$.*

In other words, similar to how we defined a function *Ancestor* in Section 5.1.1 as a self-referential link back to facets, we now define a function that unrolls the foreign relationship between facets and resources. An example of this is seen in Figure 5.2: the resource with *resource 2* as its key holds a foreign relationship with the medication that has *anti-diabetic* as its primary key.

Relating this back to the definition above, we rephrase this as: for every facet in the graph, collect their primary keys (PKs) and from the resource table, collect any primary keys where any foreign keys match the original keys (PKs). At this point, the interface is free to present the resources as needed, which consequentially allows us to model ranking and sorting schemes for resources; we leave these discussions as future work.

5.3 Requirements of Faceted Browsing Systems

Faceted browsing is a generalized concept so naturally a large variety of needs exist across different types of systems. The difference in requirements across systems directly impacts how complex the abstractions might need to be in order to fully represent a system faithfully. In the next sections, we give preliminary thoughts on each type of identifiable faceted interface before discussing in length the importance of multiple instances in the next chapter.

5.3.1 Single Instance of Facets

The simplest case is that a single faceted taxonomy drives the interface and enables the user to successfully explore and identify resources of interest. E-commerce is the most common example of this where one predetermined set of facets are given as filters for a web interface. The existing examples of published efforts in faceted browsing presented in Section 4.2.1 rely upon a single instance. This does not imply

that the interface in question is simplistic or uninteresting, but rather it implies that its requirements are simply met with a single, organized collection of facets.

5.3.2 Multiple Instances of Facets

If not a single faceted taxonomy, the interface must be driven by multiple faceted taxonomies and therefore multiple instances of our facet categories must be managed. These are not as common in the e-commerce domain, but are more common in domains where multiple existing terminologies are used to describe resources, such as the biomedical domain. For example, healthcare data contains patient and hospital or clinic visit information which can be described in turn by its elements, such as medications, procedures, diagnoses, and so on. Each of these pockets of information have multiple existing taxonomies and standards developed that could be leveraged to create a faceted taxonomy to use in an exploratory search environment. Small adjustments in implementation must be made to accommodate the extra faceted information and we outline these steps in Section 6.1.1.

Within the scope of interfaces with this type of requirement, there are two types of systems: (1) those that merge taxonomies together into a master taxonomy of smaller parts and (2) those that depend upon independent taxonomies controlling individual components of the interface. Because of their inherent design complexity, we dedicate the next chapter to these types of interfaces that require multiple taxonomies and give examples related to the biomedical domain.

Chapter 6 Requirements with Multiple Instances of Facets

We discussed previously the relationships between instances of facets and their implementation. An additional benefit of creating instances is that it enables the modeling and management of multiple taxonomies, which are typically derived from standard terminologies and consumed by interfaces as part of their exploratory search offerings. We integrate heterogeneous terminologies into our category-theoretic model of faceted browsing and show that existing terminologies and vocabularies can be reused as facets in a cohesive, interactive system. Faceted browsing systems can depend upon one or more taxonomies which outline the structure and content of the facets available for user interaction [47, 48]. Controlled vocabularies or terminologies are often externally curated and are available as a reusable resource across systems. We demonstrated previously that category theory can abstractly model faceted browsing in a way that supports the development of interfaces capable of reusing and integrating multiple models of faceted browsing. We extend this model by illustrating that terminologies can be reused and integrated as facets across systems with examples from the biomedical domain.

6.1 Examples in the Biomedical Domain

Recall that a simple example of facets for a digital library of books would be genre or publication date. The taxonomy behind the interface is either custom to the search needs of the interface or bootstrapped by a terminology familiar to those with working knowledge of the domain. In the biomedical domain, patients are often classified according to ICD10 diagnosis codes [49] in their electronic health record; as seen in Figure 6.1, the i2b2 query tool is capable of searching for patients using ICD10 codes [14] as well as other common biomedical terminologies. We will discuss i2b2 and another biomedical application in Section 6.2.

In this chapter, we integrate heterogeneous terminologies as facets into the category-theoretic model of faceted browsing so that existing and well-known terminologies can be reused in an intelligent manner. These terminologies themselves can act as a faceted taxonomy, but we also demonstrate the usefulness of modeling a terminology as a facet type. We discuss how to create instances of facets and faceted taxonomies in order for our model to interact with multiple, heterogeneous sources. We present and compare two considerations for modeling faceted browsing interfaces that utilize

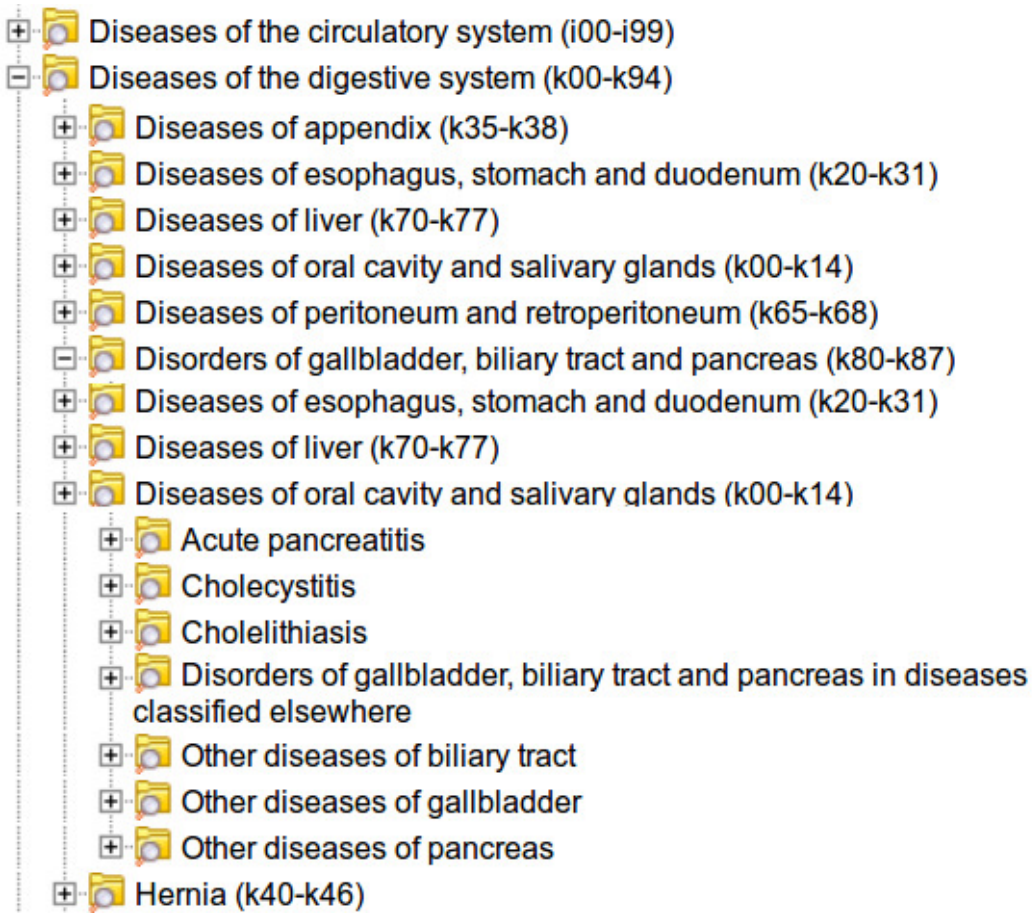


Figure 6.1: Users can select from a variety of biomedical facets within i2b2, including those from existing and well-known terminologies; a subset of the ICD10 terminology as viewed through the i2b2 query tool is shown here.

multiple terminologies: the need to merge facets together and the need for multiple focuses from different terminologies.

6.1.1 Impact on Implementation

We introduced a sample medication taxonomy in Figure 5.1 where each resource is classified using the taxonomy. In our model, we refer to resources in the general sense. The type of resource depends upon the interface: resources could be books in a digital library system, documents in an electronic health system, and so on. Note that the taxonomy in Figure 5.1 could easily be considered the facet type *medications*, which belongs to a large taxonomy (not pictured) rather than being a complete faceted taxonomy itself; either scenario is acceptable as this will depend upon the design of the faceted browsing system, which can vary. We expand this example by showing

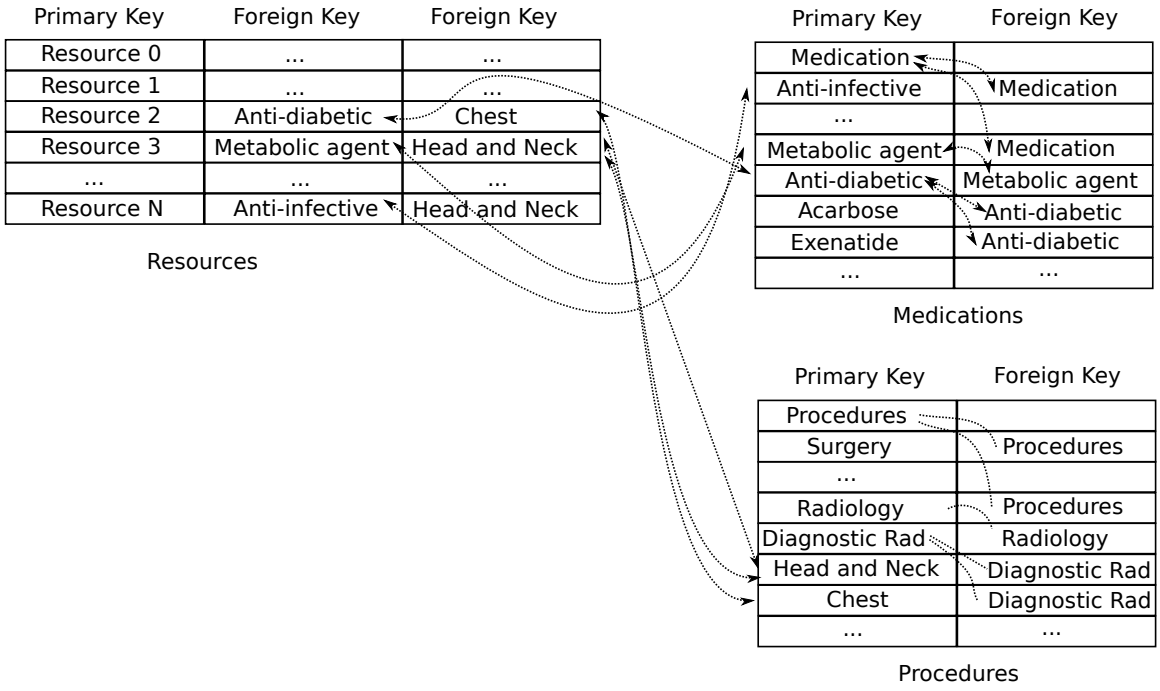


Figure 6.2: An extended example having a resource table with multiple foreign keys and corresponding tables for medications and procedures.

the impact of including an additional taxonomy to describe the resources. In this example, we can simply add another foreign key that links the resource in question to the new taxonomy; this new taxonomy requires a table to store the ancestor-descendant relationships. If there are N taxonomies, there would be N foreign keys added to the table of resources. An alternative method would be to split the resources and foreign keys into individual tables, which is described in the next section.

6.1.2 Mapping Multiple Instances to a Database

Our category-theoretic model of facet browsing uses instances which in turn require the ability to recover ancestor relationships of objects classified within a facet. In the previous chapter, we saw that this is mappable to schemas and that the facet itself requires two things: (1) a table containing ancestor links that are foreign keys to itself and (2) a table of resources and a foreign key pointing to which facet the resource is classified. If we consider multiple taxonomies, the easiest solution is to repeat the process and for each taxonomy, maintain separate tables for each facet's structure and each facet's relationship. The end result for a hypothetical requirement of having both medication and procedure taxonomies is drawn in Figure 6.3. It is

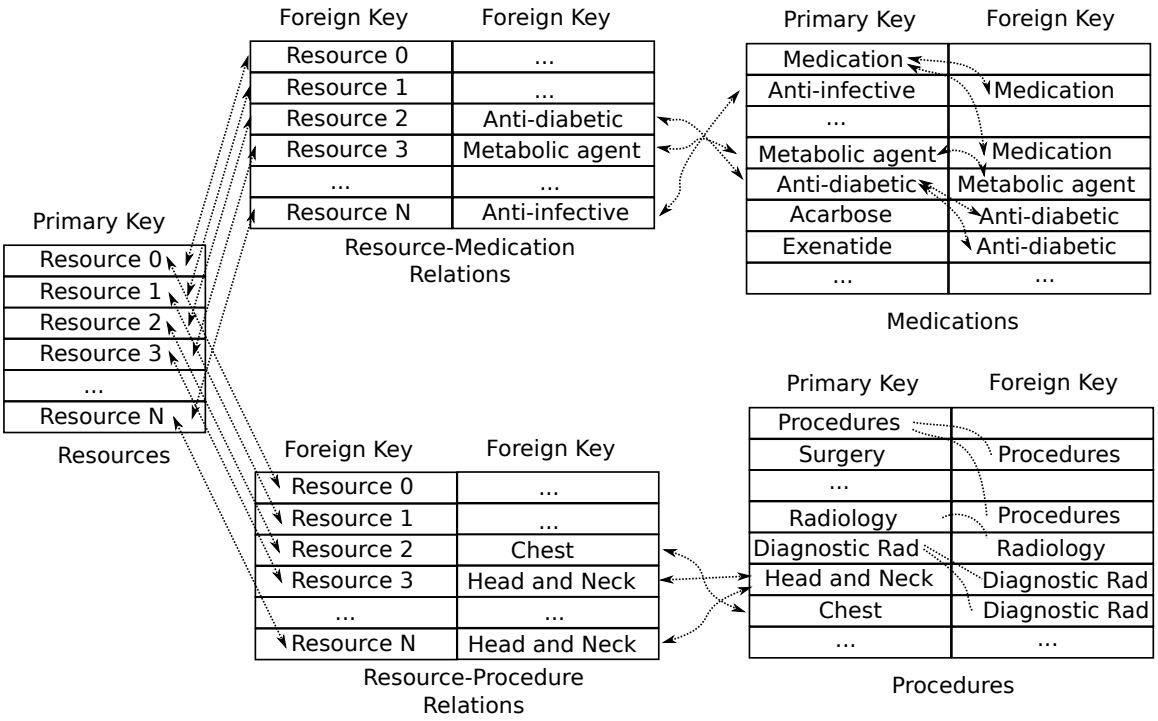


Figure 6.3: An extended example having multiple resource-relationship tables containing foreign keys that point to both resources and facets.

important to note that because multiple facets are describing the same resource, the resource in the resource-facet relationship table is now a foreign key too. This allows resources to grow and shrink with easy management of their relationships. Furthermore, it is easier to add tables than it is to add columns in most relational databases because adding columns requires migration of the old table structure into the new table structure.

6.2 Faceted Design Considerations

Faceted taxonomies are common in the biomedical domain where controlled vocabularies are curated and integrated into interfaces in order to assist in the exploration and interaction required by the system. We present two different use cases for faceted taxonomies with different requirements: one where merging heterogeneous terminologies into a single taxonomy fits the design of the interface (for example, i2b2) and one where having control over multiple independent instances of facets is desired (for example, DELVE).

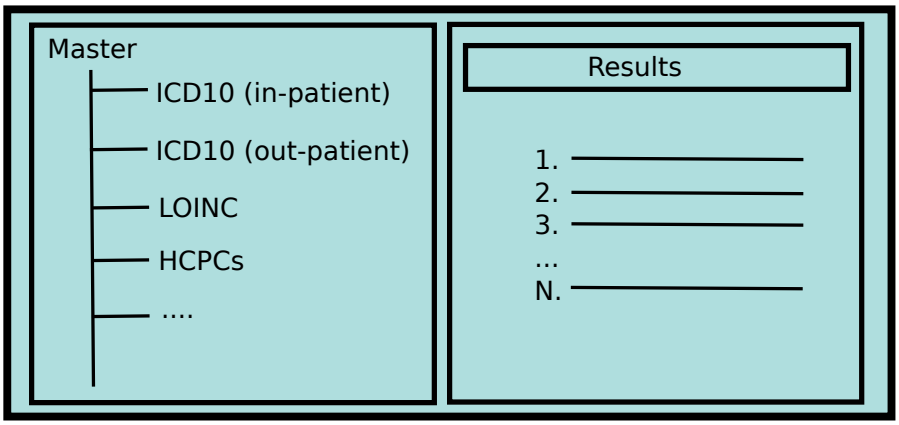


Figure 6.4: A web interface could merge multiple instances together into a master taxonomy.

6.2.1 Designing Faceted Systems

A common design for faceted systems that require multiple terminologies is to simply merge everything together into a centralized master taxonomy; this merged taxonomy is often how lightweight ontologies, discussed as one of the three foundations of facet models [1], are constructed. The merged taxonomy may or may not have multiple instances of the same terminology, depending upon what is needed for the interface. For example, in the conceptual skeleton of the interface presented in Figure 6.4, the merged taxonomy has multiple existing biomedical terminologies, including two instances of ICD10, based upon whether the resources are classified as belonging to in-patient or out-patient resources. In Section 6.2.2, we will discuss i2b2, a modern biomedical research tool that estimates patient cohort sizes by constructing Boolean queries from a merged faceted taxonomy.

Alternatively, multiple terminologies can peacefully co-exist within a single interface without being merged into a master taxonomy. In fact, it could be a pivotal design element in the interface that allows for a deeper exploratory search of the resources by enabling multiple points of faceted search. In Figure 6.5, we show a conceptual skeleton for a faceted system utilizing multiple terminologies and multiple instances of ICD10. For example, such an interface could leverage ICD10 to draw a graph of facets (i_0) and a tree of related facets (i_2) and enable the user to interactively explore resources which could be a simple list with annotations (i_1). This example is similar to the spirit of DELVE, discussed in Chapter 7, where facets are contained within and help drive visualizations.

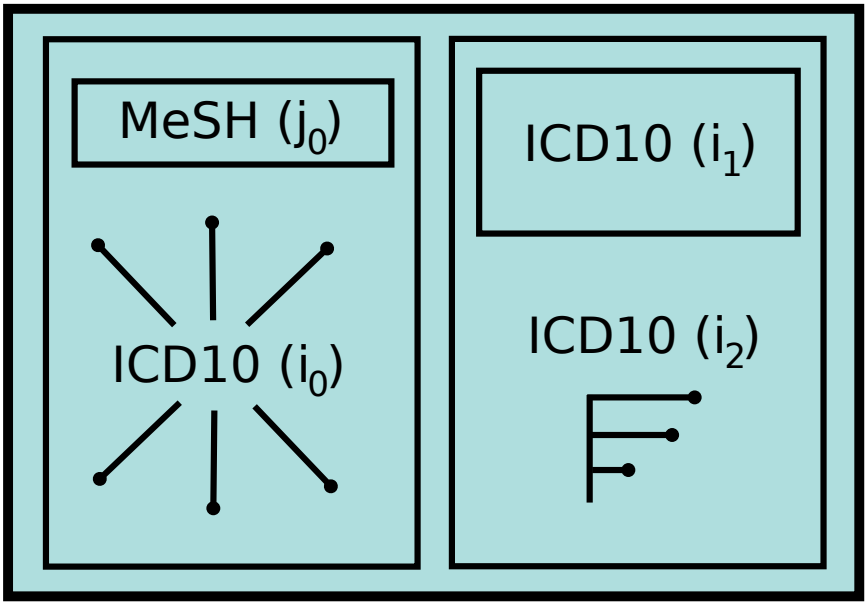


Figure 6.5: A web interface containing multiple instances of a terminology in discrete components assists interaction.

6.2.2 Interfaces with Merged Taxonomies

The i2b2 (Informatics for Integrating Biology and the Bedside) query tool allows researchers to locate patient cohorts for clinical research and clinical trial recruitment [14]; the tool itself provides a drag-and-drop method of creating Boolean queries of inclusion and exclusion criteria from a hierarchical list of facets. For example, if someone wanted to search for only female patients, they would click into the *Demographics* facet, into the *Gender* facet, and drag *Female* to the first query panel. In addition, if they wanted female diabetics, they would also navigate into the *Diagnoses* facet and drag the desired type of diabetes into the second panel. i2b2's Boolean queries are formed from having logical *or*-statements across panels and *and*-statements within a panel. With respect to the example above, if the user wanted female diabetic and hypertensive patients, they would also find the hypertension facet and drag it into the same panel having diabetes, so that the panel represents patients having either diabetes or hypertension. This Boolean construction can be continued with any number of facets from any number of terminologies.

The biomedical domain has a long history of curating and maintaining controlled vocabularies and terminologies, such as those found in the Unified Medical Language System (UMLS) [50]. The structure behind these terminologies is a rich source for building faceted browsing systems that explore resources having been classified with

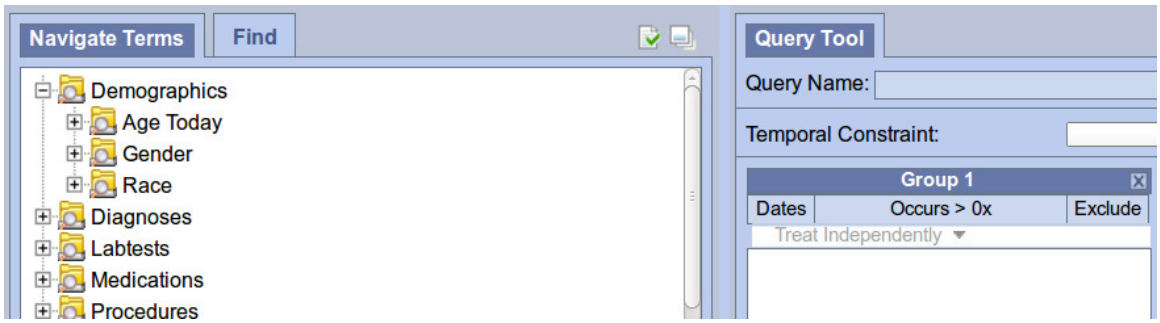


Figure 6.6: The i2b2 query tool uses drag-and-drop interaction to construct patient queries.

these standards.

In Figure 6.6, the taxonomy of a local implementation of i2b2 is partially shown; note that every facet type of a patient is compiled into a central taxonomy as part of the meta-data cell for i2b2 [14]. This means that the central taxonomy has very different concepts, such as diagnoses and laboratory procedures, residing in the same table.

Our local implementation of i2b2 uses ICD10 codes [49] for diagnoses and HCPCs codes [51] for procedures; these terminologies are externally and independently curated and made available by their creators. To i2b2, diagnosis is a facet type and ICD10 provides the organizational structure behind diagnoses, but ICD10 is a full terminology and one can consider ICD10 itself to be a faceted taxonomy for diagnoses; the use of large-scale existing terminologies in faceted browsing system blurs the line between facet types and faceted taxonomies, similar to our example and discussion of Figure 5.1. There is flexibility in our model that allows a terminology to be either a facet or a faceted taxonomy depending on what is appropriate for the interface that is being modeled. We take advantage of the notion of flexibility later in Chapter 7 when we discuss DELVE and modeling multiple instances of faceted terminologies within a single system.

Our modeling technique needs to be able to abstractly and consistently model both of these cases. In either case, the goal is encouraging the reuse of existing terminologies so that our faceted taxonomies contain accepted interoperable standards. An extension of i2b2 allows networking queries between institutions, so that one Boolean query can return counts of patients from multiple clinical sites; this would be impossible without integration of accepted biomedical terminologies into the faceted backbone of i2b2. It is important to note that i2b2 does not prescribe

what terminologies are to be used; this is a decision for local implementations, so the actual choice of terminology may vary. Standardization does not solve this problem as multiple standards exist for any given facet for health record data; modeling solutions need to be generic and adaptable to any faceted taxonomy so that further application development can proceed quickly.

Regardless of what standard biomedical terminologies are chosen to bootstrap i2b2's faceted taxonomy, it is required that they are merged together into a single taxonomy that operates the interface. This master taxonomy is where concepts are chosen and dragged into arrangeable boxes for inclusion and exclusion criteria that target specific patient populations.

6.3 Merge Operations

Supposing we have multiple instances of facets, $I_0, I_1, \ldots, I_N$, how do we satisfy the requirements of an application such as i2b2 that expects a single instance to act as a master? For example, I_0 could be medications, while I_1 could be procedures, and so on.

Each $\mathbf{Facet}_i$ category is disjoint and contains no linkage to another $\mathbf{Facet}_j$ where $i \neq j$, so we must manufacture a link. This link is a meta-facet, an organizational tool that typically aids in drawing the faceted taxonomy [4].

By design, the meta-facet must connect to the root of each facet; we can easily identify the root in our facet graph because it is the only entry with a null ancestor. Given an instance, such as I_0 above, we know that the root of I_0 is the source of an arrow $a \in A$ from $U(\mathbf{Facet}_0)$ where $Ancestor(a)$ is the empty set; we shall call this function that returns the root object $root(I_i) : A \to \mathbf{Set}$ for some instance I_i.

Definition 20. *$\boldsymbol{Facet}_M$ is a meta-facet category for categories $\boldsymbol{Facet}_0, \ldots, \boldsymbol{Facet}_N$, containing a meta-object and the roots of the others such that:*

$$Ob(\boldsymbol{Facet}_M) = M \cup root(I_0) \cup \ldots \cup root(I_N)$$

M is a meta-object sharing a relationship with every object: $Hom_{\boldsymbol{Facet}_M}(M, x)$ for each $x \in Ob(\boldsymbol{Facet}_M)$.

Figure 6.7 illustrates adding a meta-facet to join together a collection of facets; each black subtree represents a particular facet type. M is a new meta-object that must be created; the gray and dotted arrows that link this meta-object and the roots of the other facet graphs must be created as well.

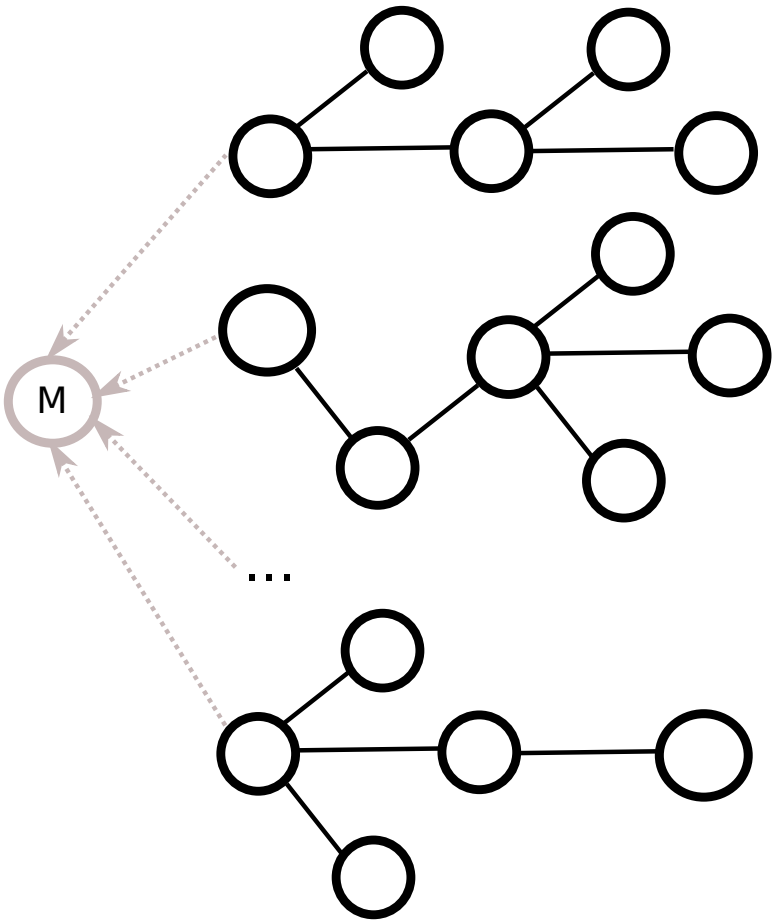


Figure 6.7: A meta-facet can assist in merging facets together by providing a common anchor point.

Let us define the union of two underlying graphs, $U(\mathbf{Facet}_i)$ and $U(\mathbf{Facet}_j)$, as the union of its constituent parts. By definition, the sets of vertices and arrows for graphs underlying two **Facet** categories, $\mathbf{Facet}_i$ and $\mathbf{Facet}_j$, are disjoint and can be merged with the union of corresponding vertices and arrows; this leaves the graph disconnected, since $\mathbf{Facet}_i$ and $\mathbf{Facet}_j$ have no object in common.

Using the root of each instance and a meta-facet, we can create a new instance connecting every other underlying graph to our meta-facet:

Definition 21. *The merger of instances $I_0, I_1, \ldots, I_N$ of categories $\boldsymbol{Facet}_0, \ldots, \boldsymbol{Facet}_N$ is a new instance I_M on $(G_U, \simeq_U)$ where:*

1. *$G_U = U(\boldsymbol{Facet}_0) \cup \cdots \cup U(\boldsymbol{Facet}_N) \cup U(\boldsymbol{Facet}_M)$. This is the union of the underlying graphs of the meta-data facet and the facets that are merging.*

2. *$\simeq_U$ is a congruence on G_U. We define this the same as in Section 5.1.1 but*

do note that the collection of paths have grown. No two paths in the merging categories conflict because the facets are disjoint by definition.

The merged instance I_M is not defined much differently than $I_0, \ldots, I_N$ in that it still maintains $(Facet, Ancestor) : F \to \mathbf{Set}$ function mappings; the only difference is that the underlying graph has changed with additional path considerations. The merge operation is simply a transformation: we are manipulating the facets into a graph and symbolically merging graphs to suit our needs. The information regarding classified resources that is embedded into each facet gets reused; only the surrounding structure changes.

6.4 Alternative to Merging Facets

We stated previously that some interfaces do not require faceted taxonomies to be merged together. In these situations, discrete components of an interface are independently controlled by a faceted taxonomy, so merging is undesirable. We give DELVE as an example of this type of interface and dedicate the next chapter to understanding how multiple components can be driven by different faceted taxonomies. We discuss DELVE from abstraction to implementation and indicate how reuse is emphasized by our model.

Chapter 7 Faceted Browsing with DELVE

In this chapter, we discuss DELVE (Document ExpLoration and Visualization Engine), our framework for developing interactive visualizations as modular Web-applications in order to assist researchers with exploratory literature search [8, 52]. In fact, our motivation for choosing category theory began when first designing DELVE; we experienced difficulty in modeling facets that are controlled by visualizations or are found within a visualization. In the case of i2b2, the design of the interface insists on merging terminologies together into a master taxonomy that directs exploration within the interface. With DELVE supporting multiple visualizations, a master taxonomy is unrealistic as each visualization potentially requires a completely different set of facets.

The goal for web-applications driven by DELVE is to better satisfy the information needs of researchers and help explore and understand the state of research in scientific literature by providing immersive visualizations that both contain facets and are driven by facets derived from the literature. We base our framework on principles from user-centered design and human-computer interaction (HCI). Preliminary evaluations demonstrate the usefulness of DELVE's techniques:

1. a clinical researcher immediately saw that her original query was inappropriate due to the frequencies displayed via generalized clouds

2. a muscle biologist quickly learned of vocabulary differences found between two disciplines that were referencing the same idea, which we feel is critical for interdisciplinary work

First, we expand upon the discussion about our motivation for building DELVE and give details on how our framework operates and how it is constructed. We also discuss how the underlying category-theoretic model of our framework relates to the end-user interface and show that it naturally encourages the additional development of reusable visualizations by emphasizing interoperability.

7.1 Why DELVE?

The rapid pace of modern biomedical research has yielded a seemingly endless supply of peer-reviewed literature that is readily available in digital libraries. Despite general ease of access, the sheer quantity of material is a barrier for experts wishing to

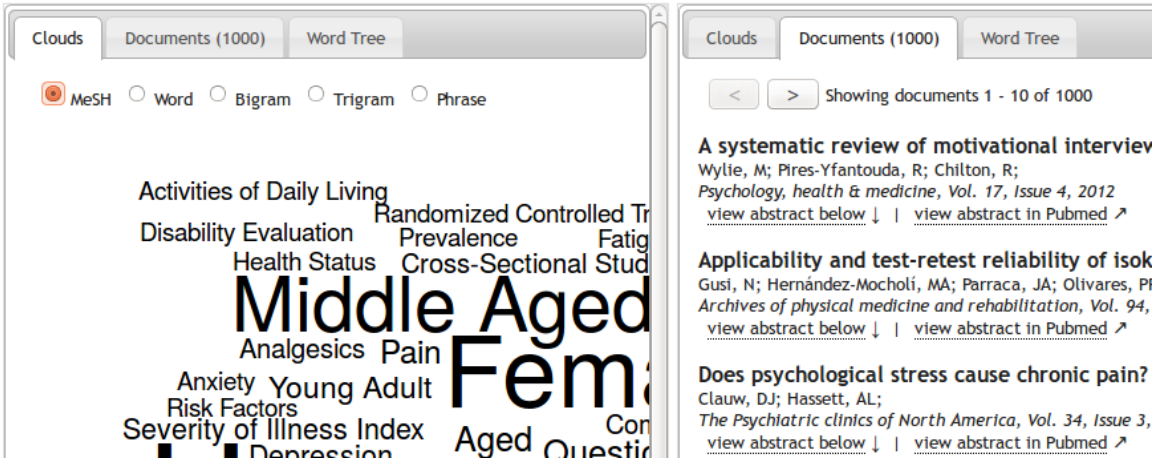


Figure 7.1: DELVE contains visualizations controlled by facets as well as visualizations that contain facets.

maintain an up-to-date understanding in their field, and this abundance of literature suggests that is not feasible to read most or all of the material in a sub-specialty [53]. Because there is an excess of information to manually review, computational tools play a pivotal role by allowing experts to ingest summarized or targeted subsections of the available literature [54]. It is not only the depth of information that is problematic but also the breadth and reach of topics that makes it crucial to understand the needs of a diverse population of researchers and to create computational tools capable of assisting in such a large variety of aims and goals [55].

Online digital libraries such as Pubmed [56] have been greatly successful in creating an online source of information for biomedical researchers and also as a source of information for computational tools to attempt to enhance or augment the literature review and dissemination experience [57]. A review of Pubmed-based applications identified twenty-eight unique systems and placed them into four general categories: ranking and search results, clustering results into topics, extracting and displaying high level semantic entities and relations, and improving search engine and retrieval experience [57]. Since the completion of this specific application review, trends indicate that applications geared toward extracting and displaying high-level semantic entities and relations [58, 59, 60, 61] outweigh those that cluster [62] or generally improve the search engine and retrieval experience [63]. All of these research initiatives are successful in their own right but lack interoperability, making the novel ideas embedded into each system difficult to reuse. Information visualization has been shown to be useful in uncovering and communicating ideas [64], yet only three applications

of the original 28 reviewed contained some type of visualization component [57]. A review of text visualization publications identified over one hundred methods of visualizing text and created a taxonomy for categorizing visualization techniques [65]; each technique contributes in some way toward communicating raw data in a more effective manner. Outside of applications that leverage Pubmed, efforts exist to visualize large collections of data and provide analytical windows on top of raw data [66, 67].

We wish to provide a framework that does not depend solely upon a single visualization technique but rather provides a suite of possible techniques that can work together in harmony. Existing online text visualization tools [68, 67] are difficult to extend either because of their closed-source nature or because they lack a natural route of integrating other visualizations that might also be helpful. We proposed DELVE (Document ExpLoration and Visualization Engine) as a general framework for interoperable and modular development of light-weight web-based applications geared toward exploring and visualizing large collections of texts in a manner that strongly supports interoperability and reuse.

7.2 The DELVE Framework

We base our framework upon fulfilling the needs of principles from user-centered design and human-computer interaction (HCI), in particular Shneiderman's information visualization mantra: overview first, filter and zoom, provide details on demand [69]. More specifically, DELVE's application programming interace (API) is capable of yielding both summarized information and the corresponding lower-level details of text documents. Filtering and zooming is supported by allowing a dynamic level of detail with each facet of information.

In Figure 7.1, a query for fibromyalgia is shown. The screen is split into two parts for this example; the abbreviated left-hand side contains a cloud [70] and the right-hand side contains a list of relevant biomedical publications. The default cloud shows the frequency of terms using the MeSH (Medical Subject Headings) vocabulary; librarians at the National Library of Medicine manually review journal articles and tag them with appropriate MeSH terms [71]. MeSH terms are hierarchically organized and are typically accurate reflections of the article's contents since they are manually assigned, making them great facet candidates. In addition to MeSH terms, we extend the general concept of world clouds [70] to unigrams, bigrams, trigrams, and common phrases.

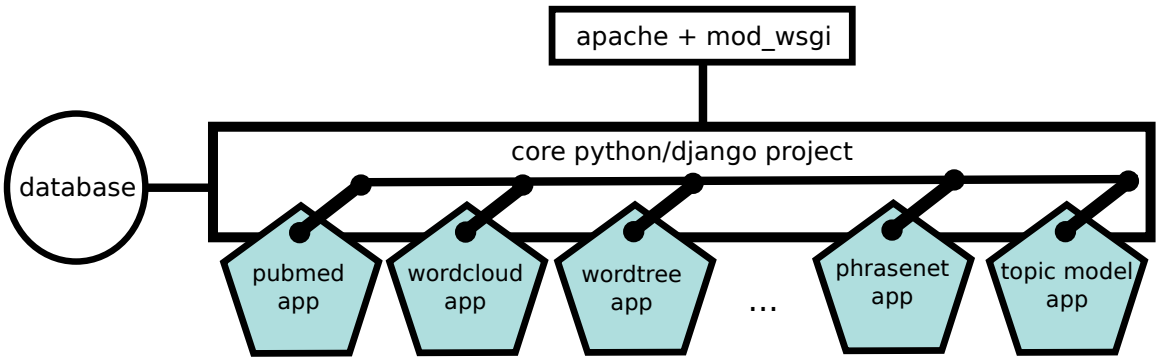


Figure 7.2: DELVE is a series of modular web applications, where each application maintains interoperability with the others via a common faceted data structure.

DELVE is implemented using the Python-based Django [72] web-development framework and is made available online as open-source software [73]. Given a Pubmed query, facets of the resulting publications are exposed via the DELVE API as files containing JSON (Javascript Object Notation) [74]. These JSON files, which carry either aggregate or detailed information per query, are used to bootstrap visualizations created with d3 (Data Driven Documents) [75]. As seen in Figure 7.2, each web-application is a modular unit that ties into a common Django model that is responsible for communicating with the raw data residing in the database and exposing the necessary JSON. In other words, each visualization is a self-contained web application and it can be placed anywhere within a larger collection of visualizations. The default layout displayed on DELVE's web site is not the only layout possible; the API allows for the arrangement of each visualization as the application's intent demands.

The seed that begins the DELVE workflow is a Pubmed query that is completely compatible with Pubmed's robust query engine, i.e. supporting tags such as *[Mesh Terms]*; the added value is that the abstracts and meta-data corresponding to these results are exposed via the DELVE API to feed directly into d3 visualizations. Because the syntax matches across systems, a researcher can directly compare the results of querying Pubmed and the results of querying DELVE. Exposing the results via JSON enables rapid development and prototyping visualizations and alternative or supplemental search experiences. Per query results, meta-data components such as publication details and authorship details are available; the text of the abstract is exposed as either (1) JSON lists of unigrams, bigrams, trigrams, and MeSH terms, or (2) partial or complete sentences depending upon need. In Figure 7.3, we visualize

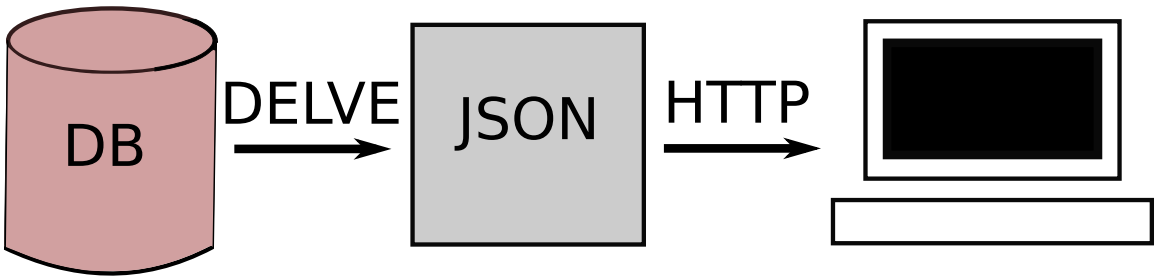


Figure 7.3: DELVE exposes the raw data stored in a relational database as JSON; the JSON is rendered as a visualization within a web page.

how data flows from the database and into a web page. The raw data is stored in the database as tables; DELVE's API transforms and exposes this data as JSON that gets transmitted through HTTP or HTTPs as part of a web page. We align our transformed data as JSON files that are directly consumable by the d3 library discussed previously.

7.3 Interoperability and Reuse

Our DELVE search tool is a special case of a faceted browsing system [1] where facets control visualizations but also where the visualizations contain facets that can be selected, such as any of the words or phrases displayed by the clouds, word trees, or phrase nets. When we first began working on DELVE, we were motivated by unifying text visualizations for exploratory search for a common purpose and implemented a selection of visualizations for a specific group of publications regarding a specific disease that interested a clinical researcher in our collaboration. Although successful as a proof of concept that encouraged us to move forward, we had great difficultly in communicating abstractly how our visualizations were actually transforming the raw data into usable modules that contribute to the larger aim of exploring text collections. At the same time, we identified that it was difficult to reuse existing work and visualizations in the same area because of the variety of theoretical foundations used to create such systems [5].

We solve the issue of communication and the issue of reusability by using the proposed abstract model of faceted browsing and use the knowledge of category theory to consume and unify other theoretical models of faceted browsing [5]. Specifically in the biomedical domain, heterogeneous terminologies play an important role in exploring and presenting data [48] and the category-theoretic model of faceted browsing

gives us precise language to describe how instances of terminologies can be used as facets.

It is easy to convert other faceted browsing systems into the category theoretical model because there exists representations of sets, graphs, lattices, and other common data structures in category theory. In other words, category theory provides a common language for describing and modeling faceted systems. Because DELVE is a framework designed to enable development, reuse is encouraged in two ways: (1) consistent abstractions imply that novel ideas and features of applications formulated in the common language can be exchanged freely and (2) these ideas can be implemented in a common manner so that they are interoperable in practice.

Facet is providing the specification for what it means to be a facet; instances of **Facet** can be created for DELVE where $I_0, I_1, I_2, \ldots, I_N$ represent N collections of objects whose data are classified according to specific relationships, which is needed in systems where more than one faceted taxonomy can be leveraged [48]. An instance of MeSH and an instance of ICD9 could be utilized in visualizations requiring both MeSH terms and diagnostic codes.

In Chapter 6, we demonstrated that the **Facet** category is structurally equivalent to **Schema**, the category-theoretic view of database schemas [48]. Both categories describe the conceptual layout that organizes information: rows/entities for databases and resources for facets. At the core of **Schema** is primary key to foreign key relationships of which we can map facet and ancestor relationships so that we can easily implement faceted browsing in a relational database system. At a minimum, this requires two tables: one with the faceted structure and one with the relationships between facets and resources. The table for the faceted structure is at a minimum a two column table with a primary key representing a facet and a self-referential foreign key representing its ancestor.

In the following section, we discuss various methods of visualizing text and give insight on how those visualizations are implemented in DELVE; we include details on their corresponding database and JSON structures. We also show how the proposed abstractions of faceted browsing inform and guide the implementations of each visualization, so the connection between the proposed facet model and each database table and JSON structure is clarified.

7.3.1 Visualizing Text

As proof of concept in planning DELVE, we specifically looked at including the following text visualizations:

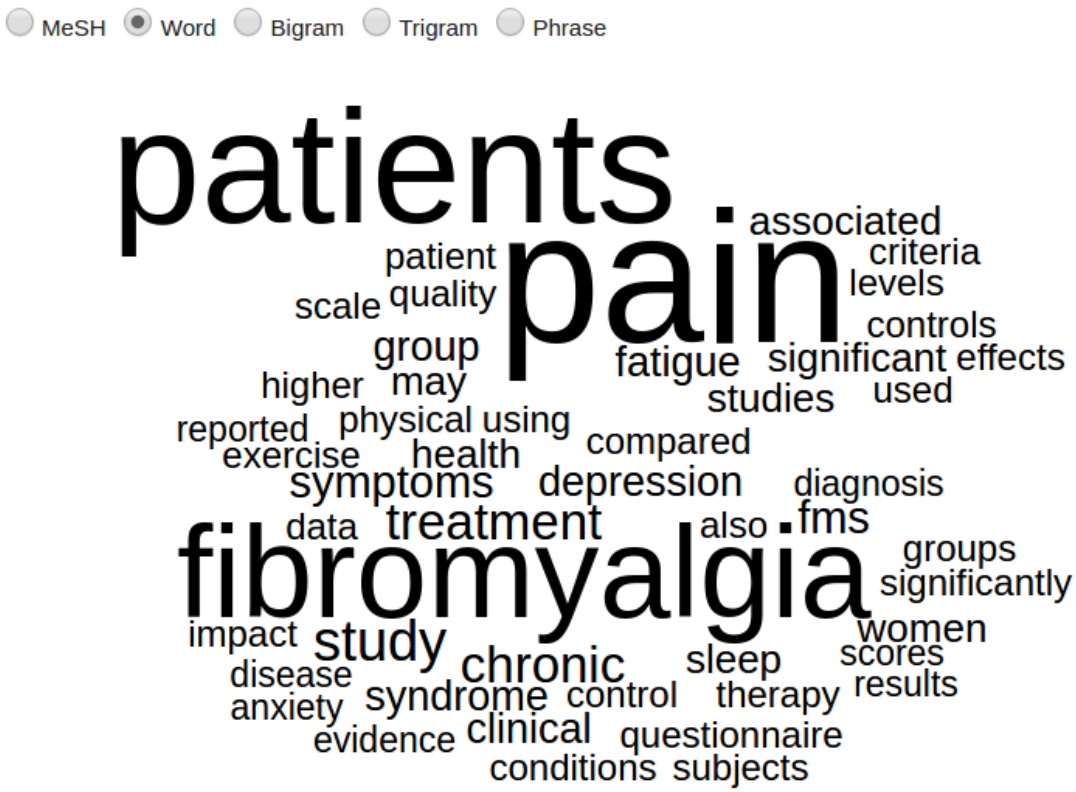


Figure 7.4: A word cloud shows frequency of unigrams extracted from the abstracts of articles. The above word cloud was generated from documents returned from a Pubmed query regarding fibromyalgia.

- clouds: visualizing relative frequency of words or phrases [70]
- word trees: visualizing sentences centered at a specific root [76]
- phrase nets: visualizing relationships centered at a specific anchor [77]

We will explore each of these visualizations in depth over the next sections and describe how they are represented by our category theoretic model of faceted browsing and how they can be easily reused.

We are also experimenting with integrating topic model analysis and visualizing the topics generated and attached to each body of text. Each of these visualizations offers something different: clouds give frequency of words or phrases, word trees give the context surrounding a word or phrase, and phrase nets give relationships between words or phrases. Many additional text visualization techniques exist [65], but the three we have selected provide a baseline for understanding the contents of

publications matching a particular query. We will discuss each of these types of text visualization in the next sections.

7.4 Creating Reusable Clouds

As seen in Figure 7.4, word clouds are visualizations where common words found within a collection of text are drawn with size relative to their frequency so that the most frequent words are drawn the largest. Clouds are not without controversy [78] but have been demonstrated to be useful in research settings [79, 80, 61, 60]. An argument against clouds is that the clutter is distracting noise and that a sorted bar graph could easily visualize the same information. This is true, but for web-applications, screen real estate is limited and only relative frequency matters for exploration; precision is not particularly valued: a frequency of x is not radically different than a frequency of $x + 1$. On the other hand, $x/2$ is a notably different frequency than x for considerably large values of x. With clouds, we densely pack frequency information into a relatively small component of the web page. A sorted bar graph containing the same information would require a considerably larger portion of the screen. Compacting this information as clouds allows us to construct interfaces with multiple visualizations to give an immersive exploratory search design. Traditional clouds are constructed from words or tags and a corresponding frequency, but we extend the basic concept of clouds for DELVE.

We generalized the concept of clouds so that they can show frequency of MeSH (Medical Subject Headings) terms [71], unigrams, bigrams, trigrams, and important phrases, which can be used to filter out or focus in on certain documents within the particular query being explored. Figure 7.5 shows an example of a MeSH cloud for a Pubmed query for fibromyalgia; for example, clicking *ankylosing spondylitis* would show those articles that were also assigned the *ankylosing spondylitis* MeSH term. Common MeSH terms such as *humans* could be pre-filtered if desired. We convert the raw text from abstracts into pairs of words and their corresponding frequency; we can then aggregate these counts per query.

In Figure 7.6, a bigram cloud shows frequencies of bigrams extracted from the abstracts of articles returned for fibromyalgia. This cloud immediately shows that *chronic pain* is the most frequently occurring two-word phrase, which is expected, but it also shows unexpected phrases such as *irritable bowel* and *sleep quality* which could be points of entry for researchers exploring the topic. Regardless of interest, the cloud acts as a method for choosing a point of focus. If a researcher clicks on

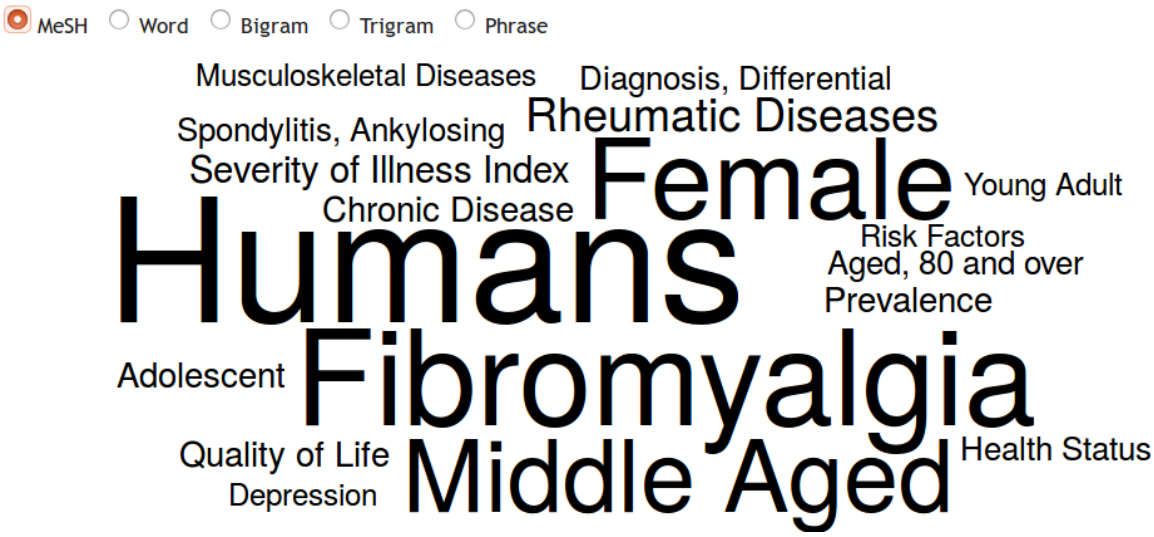


Figure 7.5: A MeSH cloud shows frequency of MeSH terms attached to a collection of articles. The above MeSH cloud was generated from documents returned from a Pubmed query regarding fibromyalgia.

irritable bowel within the cloud in Figure 7.6, only research articles containing that phrase will be listed.

Similar to bigram clouds, Figure 7.7 shows a trigram cloud for the same search on fibromyalgia. The general concept of visualizing frequency is the same but the pool of phrases visualized is different. If the trigrams and bigrams were visualized together, the frequency of the bigrams would almost always out-weight the frequency of the trigrams; for this reason, the clouds are treated independently. There are exceptions to this observation that are visible in both Figure 7.6 and Figure 7.7; the trigram *irritable bowel syndrome* occurs so frequently that both *irritable bowel* and *bowel syndrome* appear in the bigram cloud.

7.4.1 Abstracting Clouds

In this section, we discuss how to formalize generic clouds and how to implement generic clouds with DELVE. Clouds are buckets of facets, whether those facets be MeSH terms, unigrams, bigrams, or trigrams. The bucket of facets is specialized by how it is visualized according to relative frequencies of each facet found within the bucket, which we will give details on how to calculate after this section.

The concept of having a bucket of facets is identical to our category of facets known as **FacetTax**, which is simply a collection of **Facet** objects. Each **Facet** is a

Figure 7.6: A bigram cloud shows frequency of bigrams extracted from the abstracts of articles. The above bigram cloud was generated from documents returned from a Pubmed query regarding fibromyalgia.

collection of objects that are simply pointers to resources classified as belonging to that particular facet.

Each type of cloud corresponds to a taxonomy for that particular type of cloud; we can create instances of **FacetTax** objects as follows:

- I_0 - MeSH terms
- I_1 - unigrams
- I_2 - bigrams
- I_3 - trigrams

$I_0, I_1, \ldots, I_N$ denote instances as defined in Section 5.1.1 and they are constructed from the graph corresponding to each individual taxonomy. Recall that each instance

Figure 7.7: A trigram cloud shows frequency of trigrams extracted from the abstracts of articles. The above trigram cloud was generated from documents returned from a Pubmed query regarding fibromyalgia.

corresponds to two functions (one function for vertices and one function for arrows) that together map a pair consisting of the underlying graph for that particular facet and a congruence relationship on that particular graph to a simple **Set** of facets.

We can demonstrate our model using MeSH as an example. MeSH can be easily viewed as a hierarchy and the NLM provides an online tool for browsing and viewing MeSH terms [81]. A screen capture of the MeSH browser is shown in Figure 7.8; we insert into this capture a visual mapping of how our **Facet** categories could map MeSH terms. Only parts A-E of the MeSH hierarchy are shown and we only expand parts D (chemicals and drugs) and E (analytical, diagnostic and therapeutic techniques, and equipment). There are no relationships across each part of the MeSH hierarchy; for example, there are no relationships between A codes (anatomy) and B codes (organisms). This is exactly how **Facet** categories work: they are disjoint due to

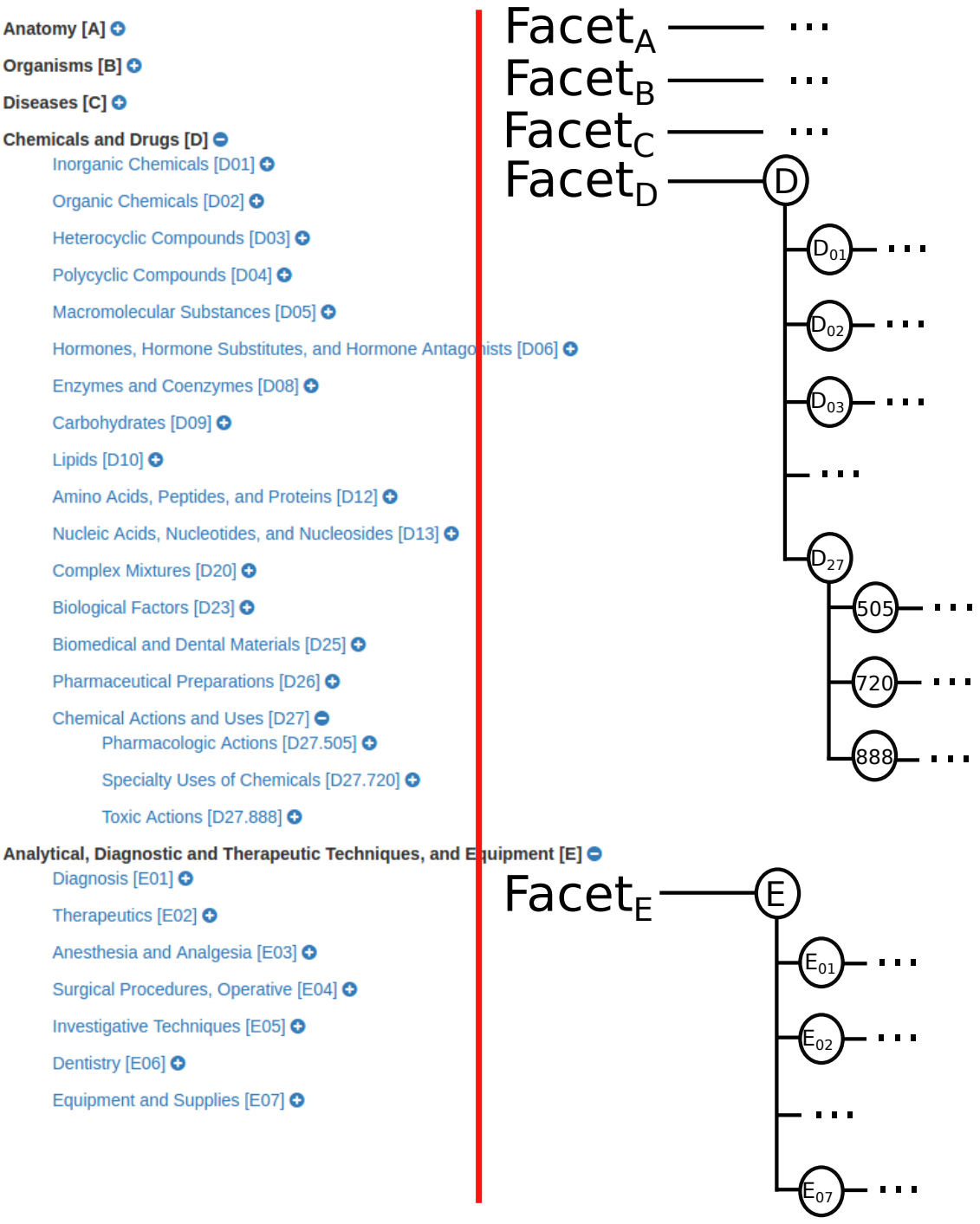


Figure 7.8: We augment a screen capture of the MeSH Browser by including how our model's abstractions would map terms from parts A-E of the MeSH hierarchy.

the nature of facet typing. Just as a book's genre has no direct relationship to its categorical price, the MeSH headings corresponding to part C (diseases) have no direct relationship with part D (chemicals and drugs). The relationships are embedded within each **Facet** category: *inorganic chemicals* (D01) and *chemical actions and uses* (D27) are both children of *chemicals and drugs* (D). There are multiple levels of MeSH's hierarchy of terms; for example, *toxic actions* (D27.888) is a child of *chemical actions and uses* (D27) which is a child of *chemicals and drugs* (D). The MeSH Browser allows one to navigate the hierarchy by clicking and expanding nodes to see ancestor-descendant relationships.

On a side note, each MeSH term maps to a concept unique identifier (CUI) in the UMLS and external relationships between concepts may exist [50]. The current DELVE prototype does not integrate with the UMLS, but there is potential for the semantic indexing of abstracts where extracted concepts can be visualized instead of lexical phrases.

7.4.2 Implementing Clouds

In Chapter 6, we demonstrated that schemas are equivalent to facets and can provide a means of implementation for the abstractions presented. There are two components necessary for implementation: (1) a table with taxonomy information and (2) a table with resources tagged according to entries in the taxonomy.

For MeSH terms, the taxonomy table requires an integer id that acts as a primary key, an ancestor integer that acts as a self-referential foreign key, and a label that represents the human readable version of the MeSH term:

```
CREATE TABLE mesh_terms
(
    id INTEGER,
    label TEXT,
    ancestor INTEGER references mesh_terms(id),
    constraint mt_pkey primary key (id)
);
```

There are additional pieces of information that can be added to this table, such as an indicator determined by the NLM specifying if this term is considered major [71], but they have been excluded for simplicity.

Additionally, the table containing resources is kept fairly simple:

```
CREATE TABLE pmid_mesh
(
    pmid INTEGER references pm_articles(pmid),
```

```
    mesh_id INTEGER references mesh_terms(id),
    constraint pm_pkey primary key (pmid, mesh_id)
);
```

The *pmid* column references an additional table hosting information about each article tagged with the MeSH term to which the *mesh-id* column points:

```
CREATE TABLE pm_articles
(
    pmid INTEGER,
    update_date TIMESTAMP,
    title TEXT,
    journ_issn TEXT,
    journ_title TEXT,
    journ_iso_abbrev TEXT,
    journ_volume TEXT,
    journ_issue TEXT,
    journ_pub_day TEXT,
    journ_pub_month TEXT,
    journ_pub_year INTEGER,
    medline_ta TEXT,
    medline_country TEXT,
    medline_nlm_id TEXT,
    auth_list_complete_yn TEXT,
    CONSTRAINT pm_articles_pkey PRIMARY KEY (pmid)
);
```

Note that this table could be further normalized by making a journal dimension, but this was not required by the example application's needs.

The tables required for constructing clouds with unigrams, bigrams, and trigrams differ slightly. Because unigrams, bigrams, and trigrams are lexical extractions from the article text, they have no embedded ancestor and descendant relationships. In this sense, they act more like a bag of words or a flat hierarchy with one level. A different type of extraction, such as concept extraction, could yield concepts from a standardized terminology [50] and structure from the terminology could be inserted into the table. Given this, the table is simple:

```
CREATE TABLE bigram_terms
(
    id INTEGER,
    label TEXT,
    constraint bt_pkey primary key (id)
);
```

An additional difference between MeSH terms and unigrams, bigrams, and trigrams is that a MeSH term can be linked to an article or not while a bigram can be linked to an article and even multiple times if it occurs more than once in the corresponding text. We store this frequency as an additional column to describe the relationship between the word and article. We give bigrams as an example but the unigram and trigram tables are virtually identical except for naming conventions:

```
CREATE TABLE pmid_bigrams
(
    pmid INTEGER references pm_articles(pmid),
    bigram_id INTEGER references bigram_terms(id),
    frequency INTEGER,
    constraint pb_pkey primary key (pmid, bigram_id)
);
```

We can also store additional information about this relationship, including what section of the text from which the word was extracted.

Formalizing Visualization Cloud Computation

The model clearly provides structure for the taxonomy and the resources, but it also provides a way to formalize computations. As a simple example, consider MeSH clouds. Recall that each object of **Facet** is a set of resources that has been classified as belonging to that facet. If I_0 is an instance of the facet type for MeSH, then we can easily generate a sequence of frequencies for objects $x_1, x_2, \ldots \in Ob(I_0)$ by simply considering their cardinality:

Definition 22. *Given an instance of a facet I_0, for $x \in Ob(I_0)$, let the frequency of an object x be defined as $freq(x) = |x|$. Let $freq(Ob(I_0))$ be a n-tuple of frequencies: $freq(Ob(I_0)) = (freq(x_0), freq(x_1), \ldots, freq(x_n))$ where $n = |Ob(I_0)|$.*

Given that we can know all of the frequencies for facets in a facet type, we are free to use that information in other calculations. There are different strategies for computing font size within a cloud, taking either the raw frequency or the square root of the frequency and mapping that to a range of font sizes [70].

One possible solution is linear scaling with an offset:

$$s = \frac{x}{max(freq(Ob(I_0)))} + \varepsilon \tag{7.1}$$

where x is a given frequency from $Ob(I_0)$ and ε is a desired offset that could enforce a minimum font-size if desired; if not, ε can be set to 0. For example, if the scale of the

font size ranges from 0 to 1 and the maximum frequency is 100, then our offset scale s given a frequency x would be calculated as $s = x/100+\varepsilon$. Looking at boundary cases, words with the maximum frequency found would scale at $1+\varepsilon$ and words with the smallest frequency possible would scale at $1/100+\varepsilon$, assuming a minimum frequency of 1.

7.4.3 JSON Structure

One of the many benefits of DELVE is that complex sources of information are exposed as JSON files in order to better support visualization. In this particular case, text from the abstracts are processed into a list of words and their corresponding frequencies. JSON is one of the default file types that can be visualized by d3, an open-source library for visualizing data within web pages [75].

In Figure 7.4, we show how a query for fibromyalgia is visualized as a cloud and emphasize that relative frequencies are distinguishable; the following shows the first three records when inspecting the associated JSON:

```
[ {    "word":"pain",
       "frequency":2962
   },
   {
       "word":"patients",
       "frequency":2511
   },
   {
       "word":"fibromyalgia",
       "frequency":1991
   },
   ...
]
```

7.4.4 Visualizing the Connection Between Abstraction and Implementation

In Figure 7.9, we illustrate how our abstract classes provide the specifications that we can instantiate within a database. For example, consider that instances of **FacetTax** are created requiring database tables corresponding to faceted taxonomies for MeSH terms and pointers to publications tagged with MeSH terms. Similar tables exist for unigrams, bigrams, and trigrams. The pointers are Pubmed identifiers (PMIDs) and point to the table of articles discussed in Section 7.4.2. The faceted tables contain

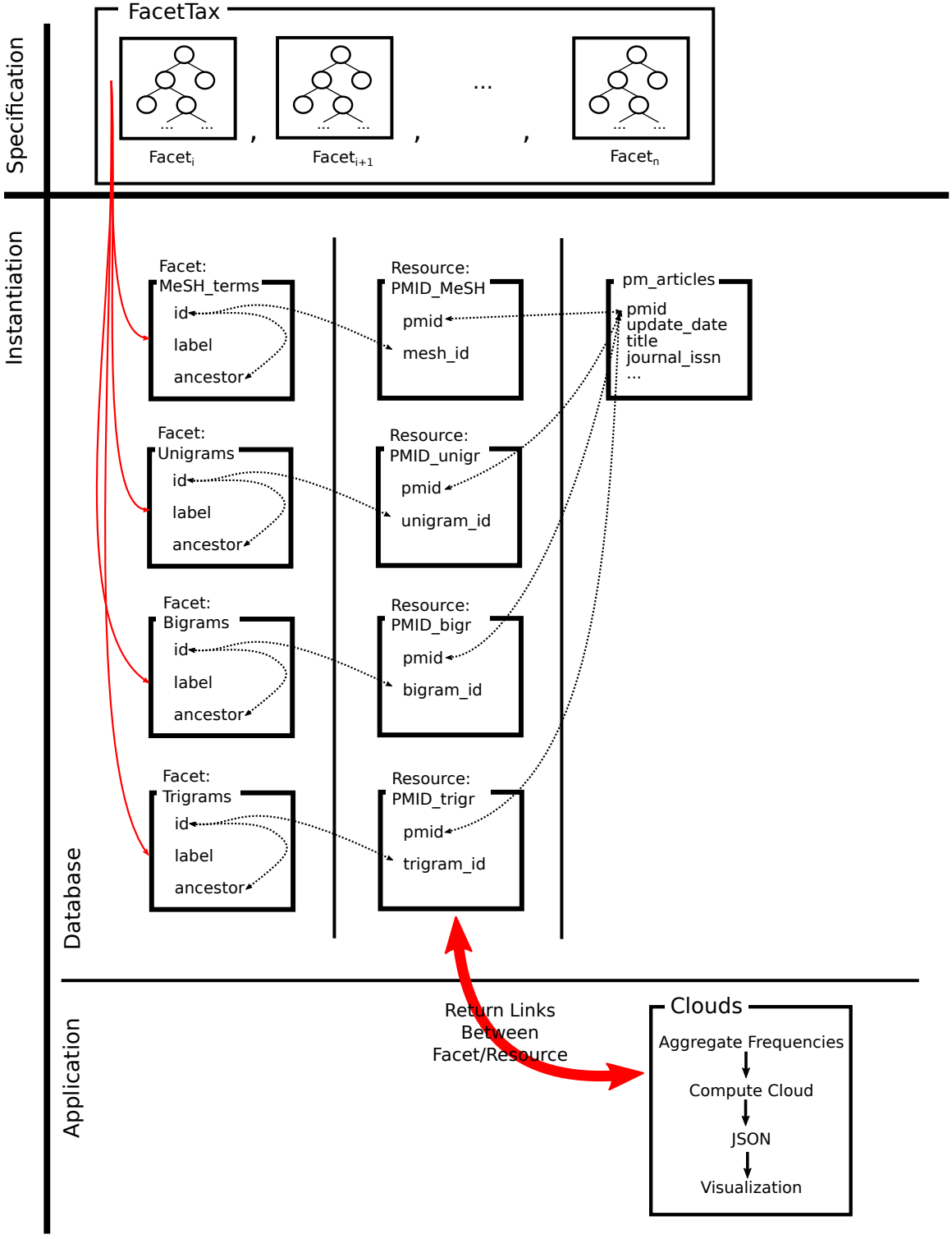


Figure 7.9: We visualize the connection between specification and instantiation and between the database layer and the application layer.

simple identifiers, labels, and a foreign key ancestor to itself and its parent. DELVE is a database-driven application, so it is important to note that DELVE ushers the data from the raw tables found in the database layer into the application layer where the cloud application aggregates frequencies, computes the cloud structure, and outputs the JSON required by the visualization library.

7.5 Creating Reusable Word Trees

Word trees are graphs that show where a chosen word or phrase appears in a body of a collection of texts [76]. Every occurrence is grouped together either by what precedes the word or by what follows it up to a configurable maximum height. For example, Figure 7.10 shows a word tree with a root word of *fear* drawn from the abstracts of documents returned from a Pubmed query on fibromyalgia. This example shows that the word *fear* is commonly followed by the word *of* which in turn is followed by a variety of fears. The document in which a chosen sentence occurs can be easily displayed after clicking a specific branch of the tree; if more than one document contains this sentence or sentence fragment, a list is presented. In this particular case, only a single article was found having *fear of* as a parent of the phrase *future, hopelessness, and mental health issues*. If clouds give frequency of certain words or phrases, word trees are a supplemental visualization that gives the context surrounding such words or phrases.

7.5.1 Abstracting and Implementing Word Trees

There are multiple ways to implement word trees. In the next section, we discuss one such method and we give details on how our category-theoretic model abstracts word trees. At this point, the visualizations need the actual text of the abstracts, so we store the text to have available for processing and computing as desired. We pre-compute what can be calculated in advance, such as delineating sentences from the raw text.

We store these abstracts as large collections of text; some of the abstracts are divided into sections such as introduction, methods, conclusions, and so on. We could potentially use this extra information in our visualizations; for example, we can show only words found in conclusions of abstracts if researchers found such delineations valuable. Additional columns can annotate the article abstracts, such as its designated NLM category [50], and we can also leverage this in our visualizations.

```
CREATE TABLE pm_abstracts
```

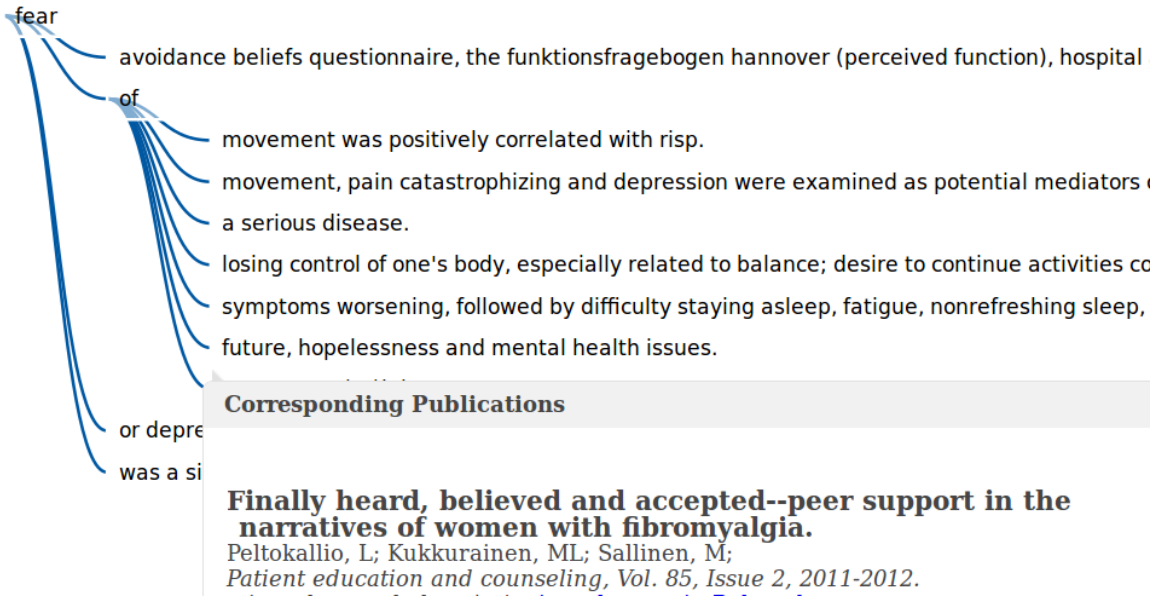


Figure 7.10: Word trees show where a chosen word or phrase appears in text. The root of this word tree is *fear* and the sentences shown correspond to documents returned from a Pubmed query regarding fibromyalgia.

```
(
    pmid INTEGER,
    section_id INTEGER,
    section_num INTEGER,
    section_label TEXT,
    section_content TEXT,
    constraint pm_akey primary key (pmid, section_id)
);
```

The actual sentences are extracted and stored in a separate table. We can feed these sentences into our algorithm that computes word trees discussed in a subsequent section below.

```
CREATE TABLE pm_sentences
(
    pmid INTEGER,
    sentence_num INTEGER,
    sentence TEXT,
    section_id INTEGER references pm_abstracts(section_id),
    constraint pm_skey primary key (pmid, sentence_num)
);
```

It is important to realize that the database attached to DELVE acts as a long-

term storage mechanism but does not directly store the computed content needed to visualize a word tree. Each word tree is specific to the pool of texts corresponding to a query and pre-computing the word tree in advance would be a waste of storage space and in most cases, simply not possible. To save space, we compute the word tree on the fly given a query and a root using the algorithms presented in the next section. This computation can be done quickly enough to maintain a live response within the web application.

In order to demonstrate how word trees are computed, it is valuable to look at the desired output that our visualization libraries will be able to consume and upon which visualizations are produced. In the next section, we detail what the desired structure of a word tree should be when considering visualization.

7.5.2 Word Trees from JSON

Word trees are structures that attempt to show how words are used in context within a collection of sentences. Every occurrence is grouped together either by what precedes the word or by what follows it up to a configurable maximum height. The data structure is recursive: a record contains a label for a word, a frequency count, a list of identifiers representing publications containing that label, and a list of children where the children are also full records having labels, counts, a list of identifiers, and their corresponding children. This JSON file is used as input for the visualization library d3 [75] which in turn renders a tree rooted at a particular phrase. The root is simply the first word in the structure that acts as a parent to every record in the tree below it. The list of publications is used to load tool tips that show additional information regarding the publication, such as its title and abstract.

For the query regarding fibromyalgia that is visualized in Figure 7.10, the following shows the first few entries of the corresponding JSON:

```
{
    "count":79,
    "pmids":[   21687553, 22935170, ...
    ],
    "name":"fibromyalgia",
    "children":[
        {
            "count":8,
            "pmids":[   22210272, 22617632, ... ],
            "name":"syndrome",
            "children":[
```

```
{
    "count":1,
    "pmids":[ 22210272 ],
    "name":"(fms) is a common chronic pain condition...",
    "children":[]
},
{
    "count":1,
    "pmids":[ 22427134],
    "name":"(fm) before and after behavioral...",
    "children":[]
},
{
    "count":1,
    "pmids":[ 22527642],
    "name":"(fms) will be presented.",
    "children":[]
},
{
    "count":1,
    "pmids":[ 21866329],
    "name":"that is acceptable to patient",
    "children":[]
  },
  ...
```


Ellipses are used to abbreviate entries. For example, the count on the root of this word tree is 79 and there must be 79 publication id numbers listed in its list of publications; to save space, we only show the first two id numbers.

In their paper introducing the concept of word trees [76], Wattenberg and Viégas focus on the utility of word trees; they detail implementation considerations such as scalability and limitations of java applets, but they do not actually give an algorithm necessary to produce word trees. We implement word trees as a list of dictionaries with the added requirement that we must track frequency of the current word at this point in the tree. We also track the identifiers attached to articles corresponding to sentences contained within the word tree, but omit this because it simply adds in tracking another list of identifiers to the algorithm. Our algorithm recursively inserts sentences into a word tree by inserting words into a list of dictionaries containing a label or name corresponding to the inserted word and a list of children (who are also dictionaries).

A basic procedure to initialize our data structure is depicted in Algorithm 7.1.

This procedure returns a dictionary containing three items: a name (label corresponding to a word), a count (frequency), and a list of children. The children will each be initialized with this procedure too.

Algorithm 7.1 Initializing a word tree node

```
1: procedure INIT_NODE(word)  ▷ Initializes fields for name, count, and children.
2:     d ← {}                 ▷ d is initialized as an empty dictionary.
3:     d{name} ← word
4:     d{count} ← 1
5:     d{children} ← [ ]      ▷ Children is initialized as an empty list.
6:     return d
```

Algorithm 7.2 will insert a sentence into a list of dictionaries. In this context, the sentence is actually a trimmed sentence that starts with the root of the tree and ends where the sentence naturally ends. For example, if the original sentence was *The cat had a hat* and the desired root was *cat*, the truncated sentence would simply be *cat had a hat*. We represent these trimmed sentences as arrays of words. The algorithm works by inserting the first word of the sentence at a specific node and recursively inserting the next word at that specific node's children. An important observation in understanding our recursive solution is that the *list* variable changes as the recursive calls continue. If a match is found in the tree, the words following the match will be inserted as children of the match. If a match is not found, the words get inserted as new nodes. The recursion ends when the array of words becomes empty after the last word of the trimmed sentence is inserted. We define what it means to insert a word at a specific node in Algorithm 7.3.

Algorithm 7.2 Inserting a sentence into a word tree

```
1: procedure INSERT_SENTENCE(list, words)
2:                                  ▷ Given a list, insert a sentence (array of words)
3:     if words is not empty then
4:         Insert_Sentence(Insert_Word(list, words[0]), words[1 : ])
```

Recall that the general structure is a list of dictionaries. In order to insert a word, we must first check to see if it is already in the list of dictionaries. If it already exists, we increment the count and return this node's children in order for the next word to be inserted at the correct position. If the word we are trying to insert is not in the list, we create a new node and append it to the list.

If there are N words to be inserted for a sentence of length N, it would require at most N recursive calls to do so. Each word insertion requires scanning a list to see

Algorithm 7.3 Inserting a word into a word tree

```
1: procedure INSERT_WORD(list, word)                ▷ Given a list, insert a word
2:     if list is not empty then
3:         for d in list do
4:             if d{name} == word then
5:                 d{count} ← d{count} + 1
6:                 return d{children}
7:     temp ←Init_Node(word)    ▷ If this word was not found, initialize new node.
8:     list.append(temp)
9:     return temp{children}
```

if it is already there or not; the length of this list is bound by how many sentences correspond to the abstracts related to a given query. For example, given a collection of M sentences, if each sentence is unique after the chosen root, then the root will have M children. Given this, we can expect the act of inserting a sentence to exhibit $\mathcal{O}(M * N)$ behavior but note that in practice $M > N$ unless the query has returned very few abstracts and as a consequence very few sentences. The utility of word trees increases as the number of sentences increases because the user is exposed to more data. If we consider the act of inserting all M sentences, we need M calls that cost $\mathcal{O}(M * N)$, each resulting in $\mathcal{O}(M^2 * N)$ behavior.

7.5.3 Visualizing the Connection Between Abstraction and Implementation

In Figure 7.11, we show the impact of including word trees on DELVE's design. The same MeSH terms, unigrams, bigrams, and trigrams are reused to act as anchors into sentences and in this context are used as roots of the word trees. To enable this, we must include the text of the abstracts at a minimum; because response time is crucial in an exploratory search system like DELVE, we pre-compute the extraction of sentences from the abstracts and maintain links between abstracts and sentences. Globally, we have links between the cloud types of MeSH terms, unigrams, bigrams, and trigrams and articles; therefore, we also have links between the cloud types and the sentences. In other words, given a specific type of facet, we can return sentences containing that facet.

The MeSH term, unigram, bigram, or trigram becomes the root of the word tree which in turn is used to split the linked sentences. The word tree is computed and the resulting JSON file is used to produce the necessary visualization. When an entry in a cloud is selected in DELVE, that entry is chosen as the selected focus of that

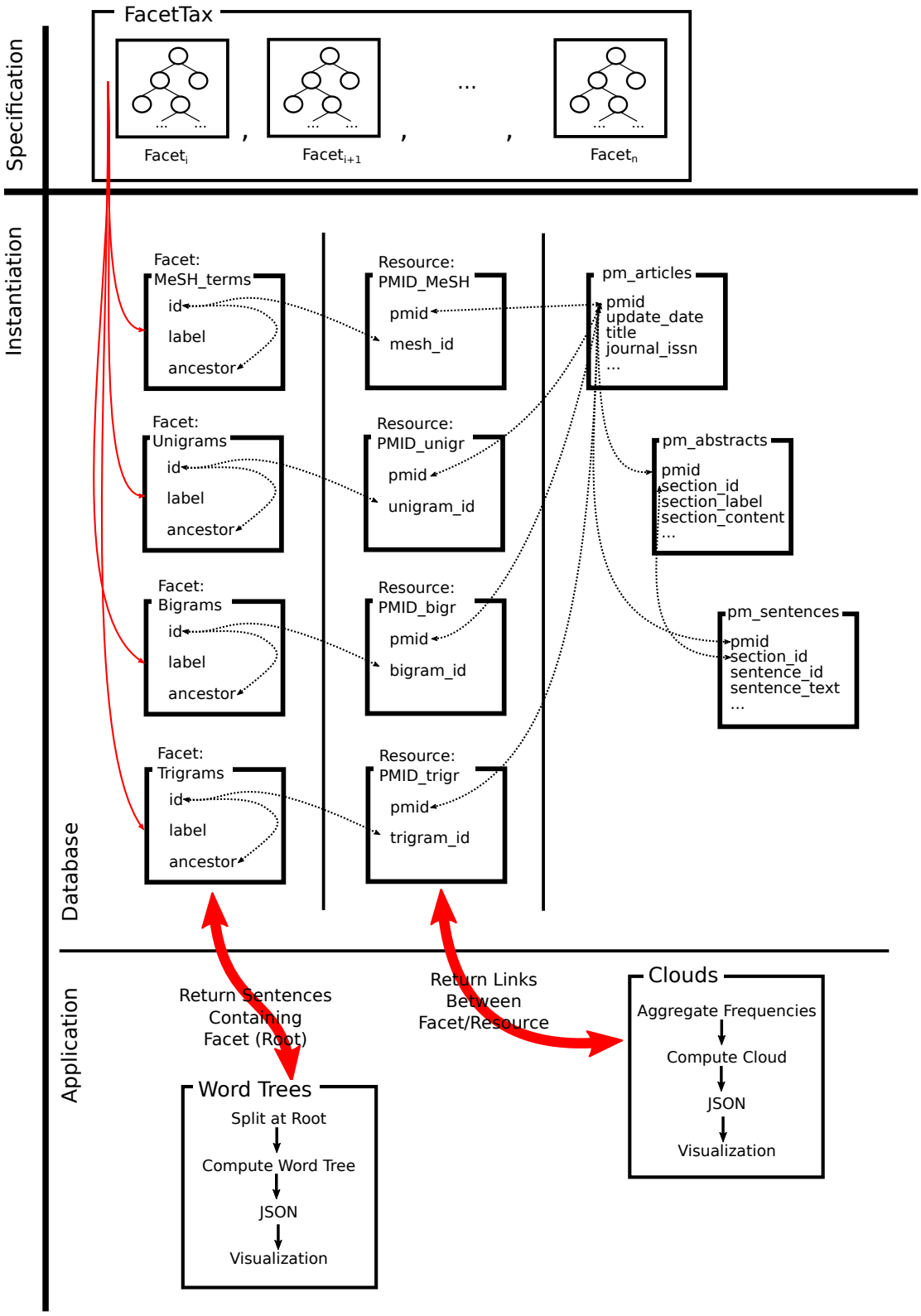


Figure 7.11: We visualize the impact of including word trees into DELVE's design and visualize the connection between the database and application layers.

query. The default root of the word tree is the selected focus. With this, we have facets found within visualization of clouds that can control word trees rooted at a certain facet.

7.6 Creating Reusable Phrase Nets

In terms of visualizing text and information found within the text, phrase nets [77] could display local co-occurrences between words or concepts in order to give insight on relations embedded into the corresponding documents. We have not implemented phrase nets, but we have prototyped how they might relate to and interact with DELVE. For example, consider a phrase net focusing on the word *and*; phrases of the form *x and y* would be drawn as $x \rightarrow y$ where (1) the direction of the arrow indicates word order and (2) the size of the arrow indicates frequency of this phrase occurring within the pool of documents returned by this query. Additionally, the color of the words can indicate individual frequency in order to show relationships between frequent and less frequent occurrences; darker coloring indicates a more frequently occurring word. In this section, we give examples of phrase nets centered around different anchor words and discuss issues that must be overcome in order to increase their utility.

When aggregating data across a large number of resources, the largest obstacle becomes the amount of unique phrases that commonly occur. Figure 7.12 shows the raw result of computing a phrase net given our example DELVE query on *fibromyalgia* and given the anchor word *and*. Heavily occurring phrases such as *pain and sleep* and *fibromyalgia and rheumatoid* are distinguishable, but the less frequently occurring phrases appear muddied. To combat the problem of visual clutter, filters need to be added to selectively display phrases.

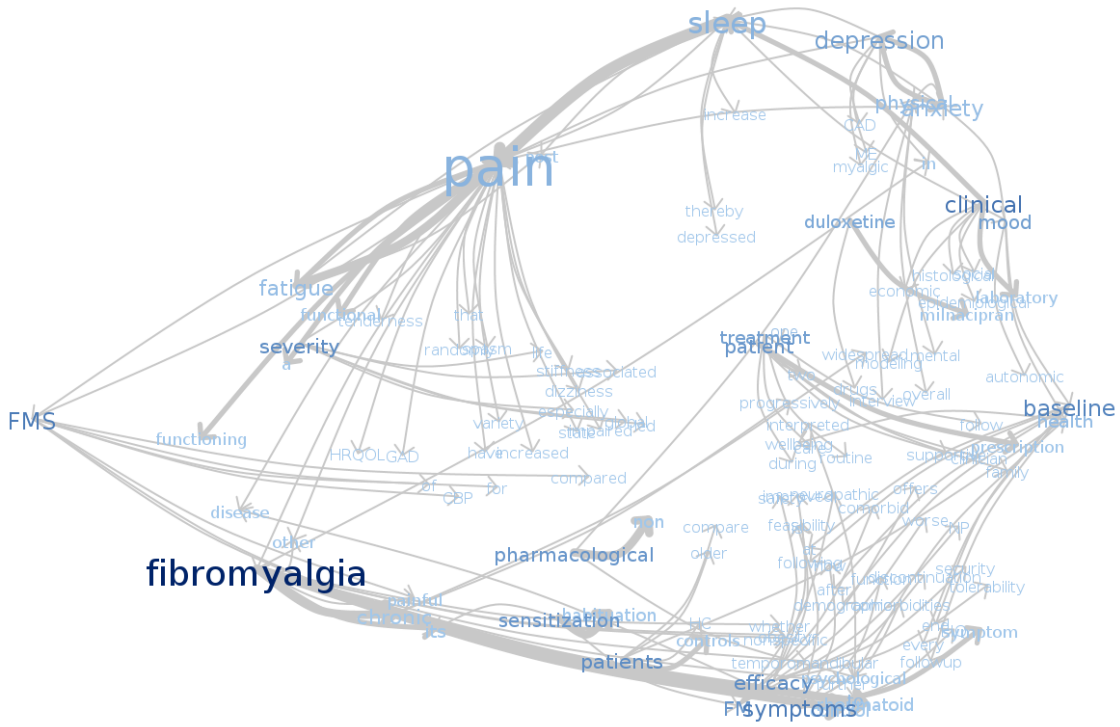


Figure 7.12: An unfiltered phrase net for a query on fibromyalgia anchored around the word *and* demonstrates the potential clutter caused by having many distinct yet frequent phrases.

We show a filtered phrase net in Figure 7.13 that uses the same data set but has been truncated only to show the most common phrases by frequency. In the filtered version, it is easier to see conjunctive relationships that frequent words such as *pain* share with less frequent words such as *spasm*, *tenderness*, and *dizziness*. The co-occurrence of words implies a relationship that the related abstracts should reflect. With DELVE, we would desire the ability to choose a phrase and see a list of its related abstracts to better support exploratory search.

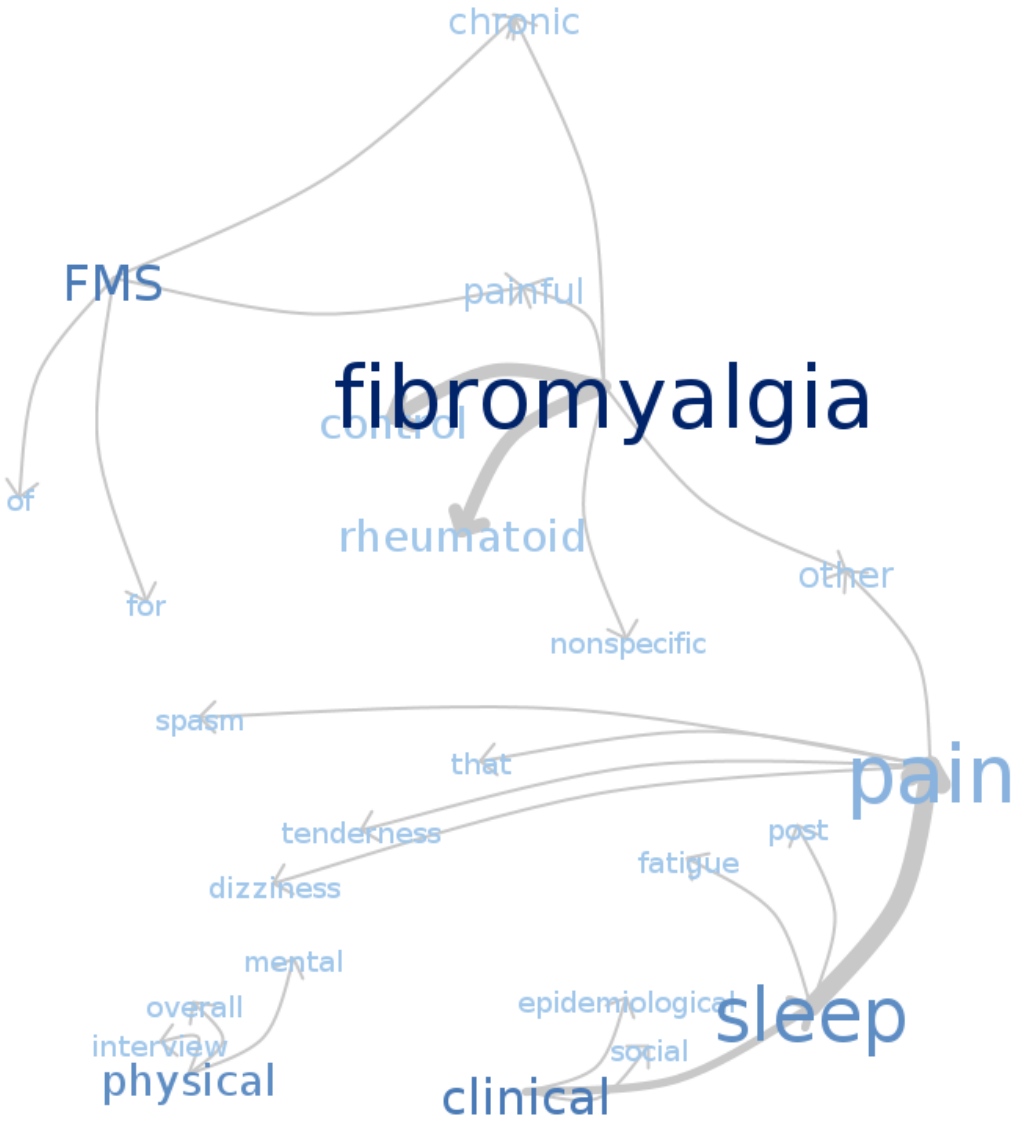


Figure 7.13: A filtered phrase net shows phrases anchored with the word *and* shows the most common conjunctions.

Phrase nets anchored at the word *and* are popular because they imply a conjunctive relationship between words, but it is also helpful to view disjunctions. In Figure 7.14, a filtered phrase net for a search on fibromyalgia anchored around the word *or* is shown. The figure shows that *prevention or treatment* is the most commonly occurring disjunction, which is not surprising given that it is a pain disorder. It is also apparent that *depression or anxiety*, *habituation or sensitization*, and *activities or environments* occur frequently. These are all points of entry for exploratory

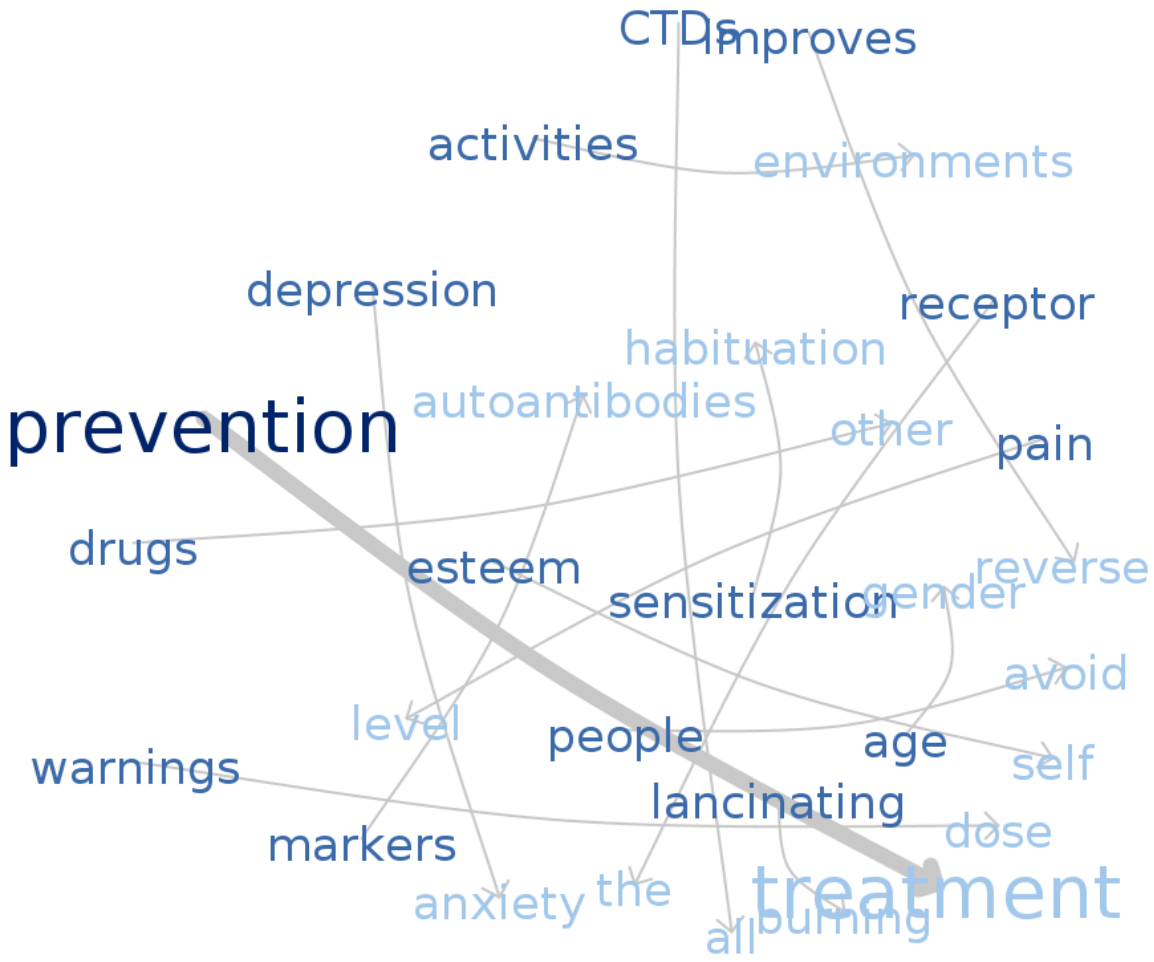


Figure 7.14: A phrase net for a query on fibromyalgia anchored around the word *or* shows the most common disjunctions.

search on the given topic. The user should be able to drill-down and investigate why *depression or anxiety* occurs so frequently to understand the context of the phrase. Either depression or anxiety could be entered as the root of a word tree and the researcher could explore the context of these phrases in the scope of a word tree.

Any word or phrase is a valid anchor for phrase nets. We have shown conjunctive and disjunctive phrases because they are popular, but there are no limitations on what can server as a valid anchor. In Figure 7.15, we show a phrase net using the same fibromyalgia example query but this time it is anchored around the word *increases*. This type of phrase net is meant to be thought provoking and meant to illicit further exploration of the data to fully understand the context of the phrases displayed. For example, the phrase *conflict increases symptoms* suggests that some type of conflict

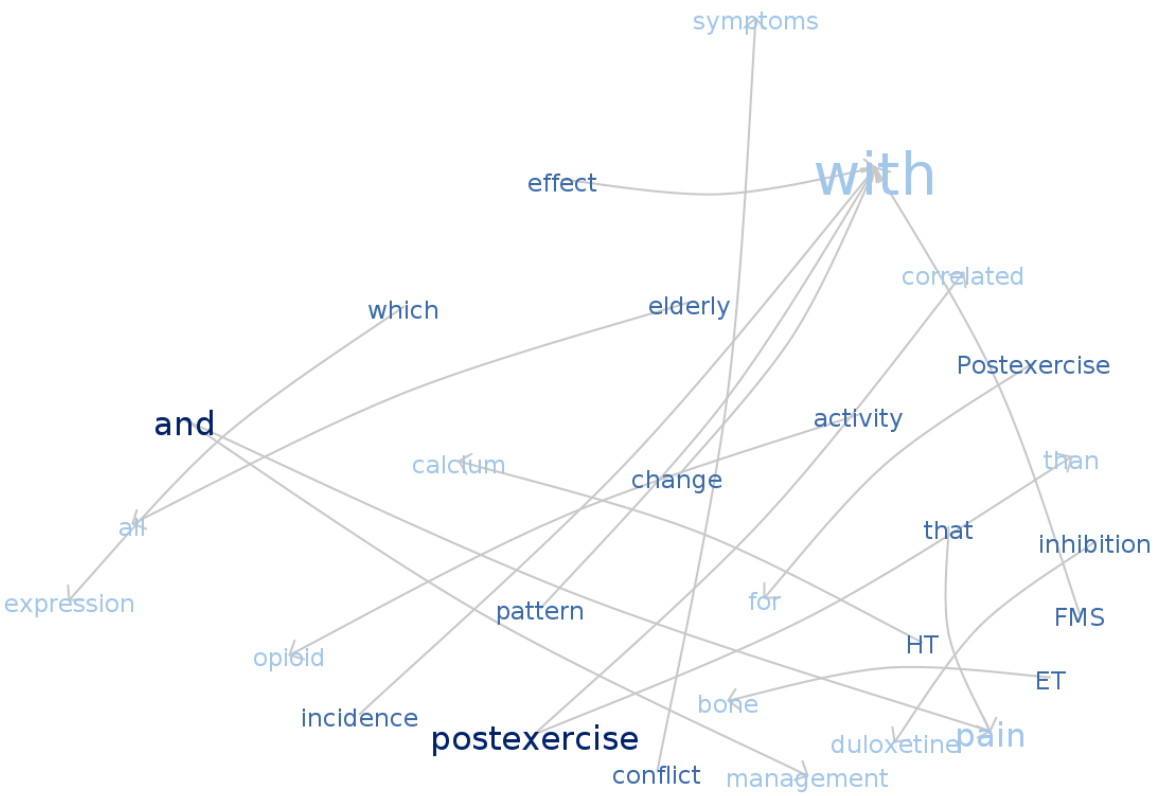


Figure 7.15: A phrase net for a query on fibromyalgia anchored around the word *increases* shows the most common words that indicate something is causing an upward trend.

results in fibromyalgia symptoms increasing; further study could illuminate why this observation might be true. For phrase nets of this nature, it is often desirable not to filter out stop words. For example, the phrase *and increases pain* occurs frequently. One could root a word tree at *and increases pain* to see the context of these statements and attempt to understand what exactly does increase pain. DELVE allows researchers to view abstracts of articles directly from the visualizations as tool tips as discussed later in Section 7.9.

7.6.1 Visualizing the Connection Between Abstraction and Implementation

In Figure 7.16, we show the potential impact of including phrase nets on DELVE's design. Each member of a phrase of the form *(a b c)* is a facet and the visualization is conveying frequency of two facets linked together by a common anchor. We envision

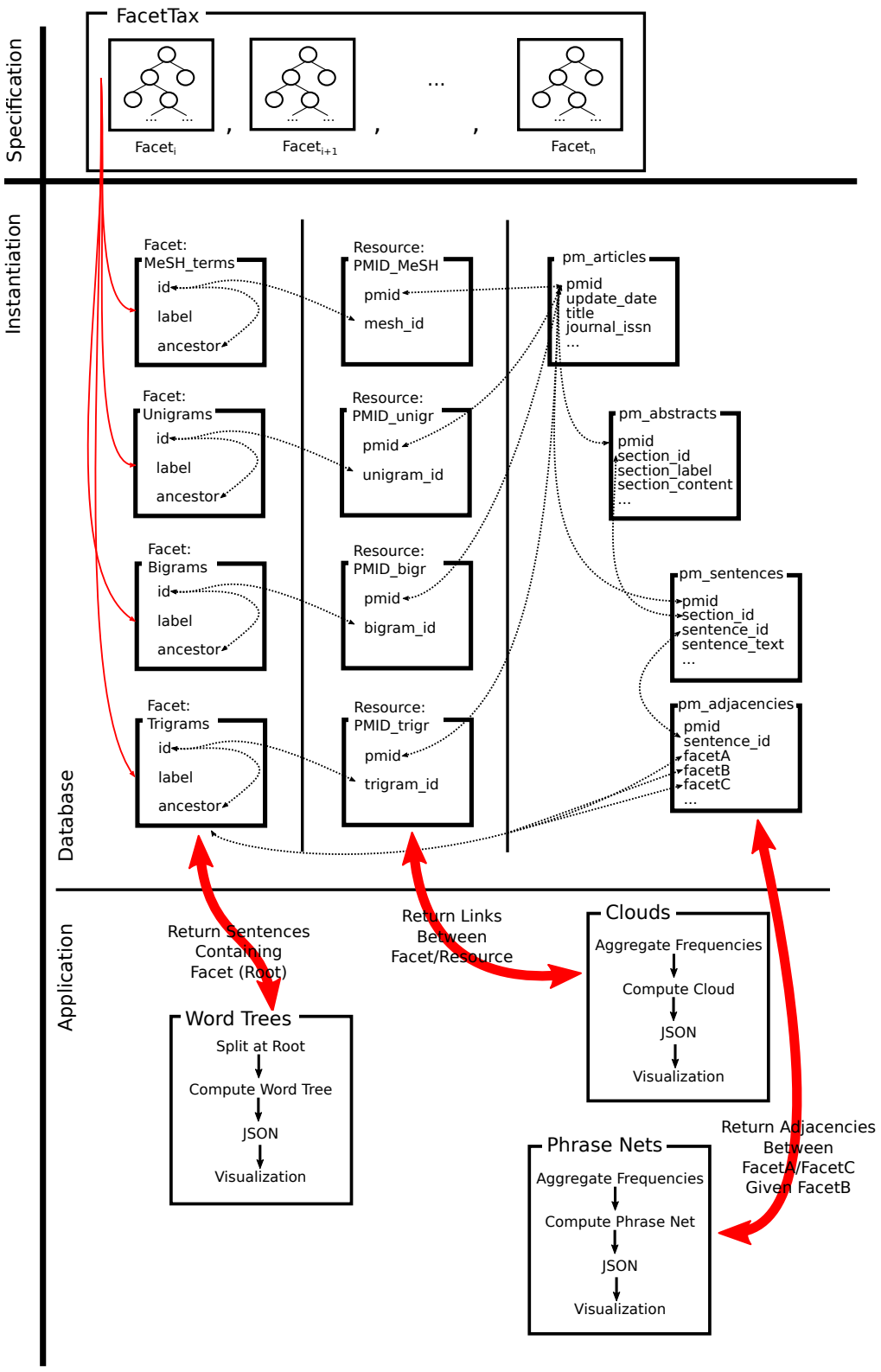


Figure 7.16: We show the impact of adding phrase nets into DELVE's design and visualize how the database and application layers interact.

that phrase nets could interact with the selected focus of a query much like word trees do. For example, if a user selects an entry in a word cloud, that selected entry would act as the focus of the word tree and the anchor of the phrase net. Alternatively, it might be helpful to include a radio-button list of very common anchors, such as conjunctive and disjunctive words.

Because phrase nets visualize phrases of the form *(a b c)*, we need some way of recovering these adjacent word relationships within our database. One potential route is to store the position of terms within the sentences found within the abstracts and for a given anchor, return the anchor at position i and its adjacent neighbors at positions $(i-1)$ and $(i+1)$. Because response time is crucial given that DELVE is a web application, we can pre-compute these phrases and store them in a table; for convenience, we can store links between the phrases and the sentences that they occur in for later use. In Figure 7.16, we call this table *pm_adjacencies*.

Recall that phrase nets can be constructed from an anchor selected within any type of cloud, whether it be MeSH, unigrams, bigrams, or trigrams. For example, if we consider constructing a phrase net from a unigram b selected from a unigram cloud, we need access to data of the form *(a b c)* where a, b, and c are all unigrams that sequentially co-occur within an abstract. Specifically for handling the data needs of unigram phrase nets, we propose the following definition:

```
CREATE TABLE pm_adjacencies
(
    pmid INTEGER references pm_articles(pmid),
    sentence_id INTEGER references pm_sentences(sentence_id),
    unigramA INTEGER references unigram_terms(id),
    unigramB INTEGER references unigram_terms(id),
    unigramC INTEGER references unigram_terms(id),
    constraint pb_pkey primary key (pmid, sentence_id,
        unigramA, unigramB, unigramC)
);
```

In other words, each record is a unique occurrence of a phrase of the form *(a b c)* for a given Pubmed article and a given sentence within its abstract. Each a, b, and c are usable unigrams and can be found in other DELVE visualizations. The goal is to streamline the preparation of the data required to construct and visualize a phrase net for a given query and a given anchor.

The application layer receives adjacencies for a given facet. The adjacencies are aggregated and the phrase net is computed; the resulting JSON is sent to the visualization library.

7.6.2 Other Visualization Candidates

Additionally, we considered integrating topic model analysis in order for computed topics to be visualized across sets of documents. Semantic knowledge bases also offer an additional source of information that could be visualized and presented as part of the suite, as later discussed in Section 7.9.

7.7 Developing with the DELVE Framework

The DELVE API is easily extended if a data element is not already exposed via JSON for consumption. If a data element was missing, one would need only to initialize the application within the Django framework and add the view logic that produces the JSON from the raw data so that it gets exposed per query at a chosen URL. For example, suppose publication date was not already available but we wish to produce a histogram of publication dates for articles corresponding to any given Pubmed query. The views in this new component would take the raw data and transform it into the JSON required by the application. Specifically, the JSON:

```
[{"total": 1, "journ_pub_year":2006},
 {"total": 1, "journ_pub_year":2007},
 {"total": 1, "journ_pub_year":2008},
 {"total": 2, "journ_pub_year":2010},
 {"total":27, "journ_pub_year":2011},
 {"total":55, "journ_pub_year":2012},
 {"total": 3, "journ_pub_year":2013}]
```

corresponds to the histogram in Figure 7.17 and is a great example of how basic histograms can imply much about trends within a research area. In other words, the presentation or visualization layer is strategically separated from the view layer that is responsible for deciding what data elements from the model should be exposed.

7.8 Driving Exploratory Search with Visualizations

The visualizations discussed in the previous two sections are good examples of taking well-known text visualizations such as word clouds and word trees and turning them into modular applications. By themselves, these applications serve a very specific purpose: word clouds provide frequency and word trees provide context. Together, these applications can join cohesively into a single interface designed to take the Pubmed

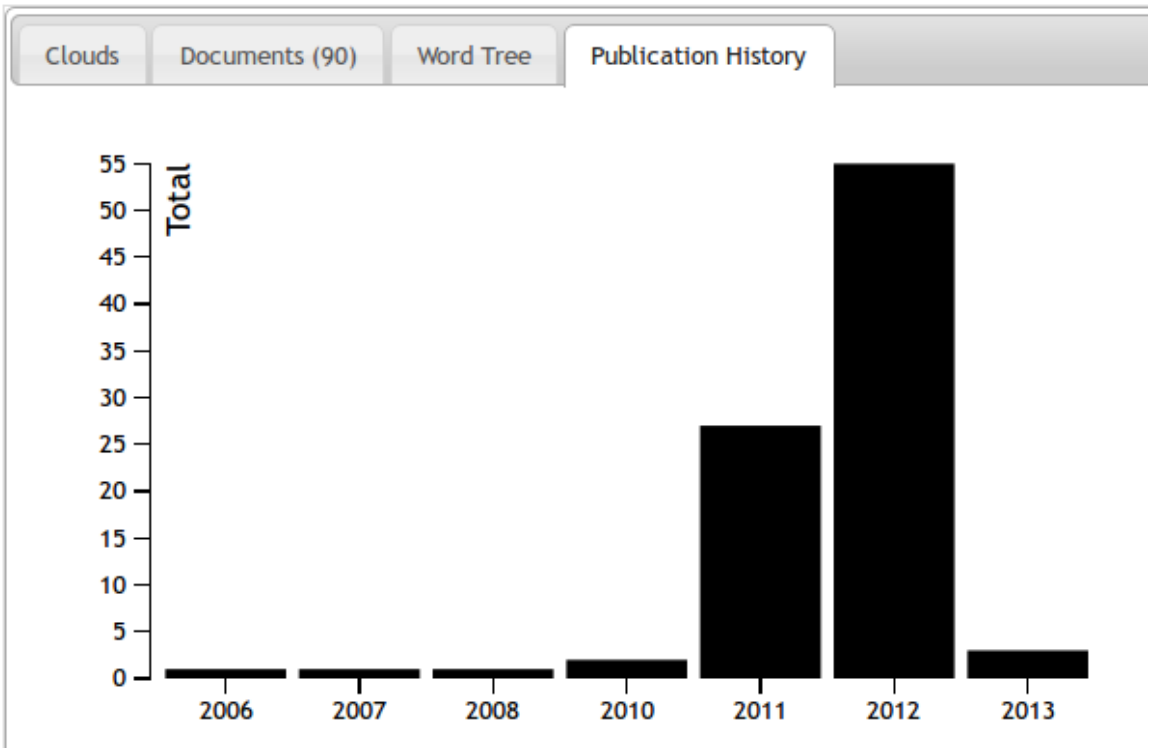


Figure 7.17: A basic histogram of publication quickly reveals temporal trends in topic popularity.

query results and immerse a researcher with a visual index of relevant publications. At this point, the visualizations generated with DELVE can either stand alone as they are or can be incorporated into a larger web application. Because we want to augment the researcher's exploratory search with additional information that would otherwise be difficult to see via traditional search methods, a centralized search portal that integrates all visualizations together is desirable.

We implemented a search tool using DELVE that offers configurable window panes of visualizations for a given Pubmed query; a sample session for a query for *fibromyalgia* and *rheumatoid arthritis* is shown in Figure 7.18-A. The general workflow consists of a Pubmed query returning associated documents that act as input for the visualization suite. Individual panes house each visualization in a separate tab. As seen in Figure 7.18, the layout of the panes is configurable so that the user can arrange the offered visualizations as they deem most effective.

For example, in Figure 7.18-D, four different types of clouds are displayed at once: MeSH, unigram, bigram, and trigram; each cloud is distinct in the list of possible words or phrases that could be displayed and that could align with what the researcher

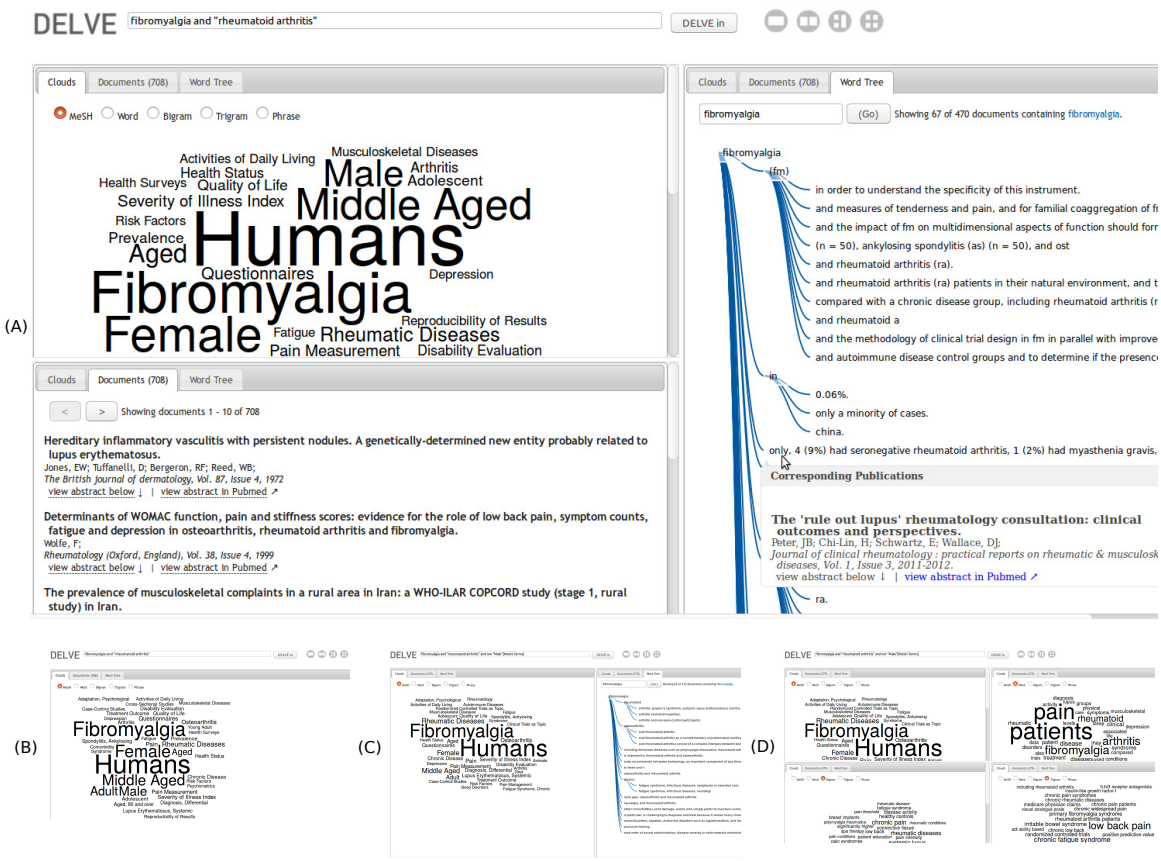


Figure 7.18: Our prototype combines different DELVE applications into a single user experience by providing integrated visualizations that collectively enable a researcher to explore collections of texts. Both the number of panes and the layout is configurable by the user in DELVE: (A) shows a sample three-pane view with clouds, word trees, and a document list, (B) shows a simple one-pane view that shows only a cloud, (C) adds a pane to show word trees, and (D) shows that different types of clouds can be viewed at once.

is seeking. For example, suppose that a researcher is interested in *persistent depressive disorder*; this phrase is not available as a MeSH term and articles are usually assigned an older variant *dysthymic disorder* MeSH term, yet DELVE provides trigrams as an alternative to MeSH terms and *persistent depressive disorder* can occur and match the intentions of the researcher.

The visualizations are linked together so that an interaction in one pane has consequences in another. The most useful interaction implemented is the ability to focus on a word, bigram, trigram, or MeSH term; focusing selects only those articles that either contain the chosen entry or in the case of MeSH terms, only those articles that were assigned that particular MeSH term. The visualizations operate on a set of

documents; if the set of documents is manipulated by the interface with an action like focusing, the visualization is updated. Since the visualization is blind to the process, the interface is free to provide any type of interactivity that filters or zooms the data.

7.9 Evaluation and Design

It is known that users struggle to successfully refine queries in search-based systems simply from looking at the number of attempts per query [82]. Because most search systems return a limited list of top search results, there is routinely not enough information presented to determine if a query was incorrect or sub-optimal, outside of the extreme case where all results appear unrelated. DELVE attempts to correct this and make query refinement easier by providing clouds that clearly display frequency of terms or phrases for all documents being returned by this query. This gives the user feedback pertaining to the entire body of documents being returned rather than only the first n-publications for those systems that return a ranked top-n list.

Scenario: Sub-optimal Search Strategy

For example, suppose that a researcher is interested in *chronic fatigue syndrome* and its relationship with *functional somatic syndromes*. The phrase *functional somatic syndromes* is not available as a MeSH term and articles are usually assigned a more general *somatoform disorders* MeSH term, yet DELVE provides trigrams as an alternative to MeSH terms and *functional somatic syndromes* occurs in high frequency, as seen in the clouds A1 and A2 from Figure 7.19. Without multiple lenses to inspect summarized and aggregated data, it may be difficult for the researcher to reconcile what his or her needs are against what the machine reports and understands. Surprisingly, a search for only *functional somatic syndromes* yields a significant number of results for *chronic fatigue syndrome*, as seen in clouds B1 and B2 from Figure 7.19. This implies that the researcher's search strategy plays a large role in how successful they may be in finding relevant articles; to combat against a user's unintentionally sub-optimal search strategy, multiple lenses placed over the data potentially compensate for weaknesses in any single lens. For example, trigram clouds usually return phrases that do not overlap MeSH terms and may match what a researcher desired.

Scenario: Poisoned Queries

In our preliminary evaluation of our DELVE prototype, a clinical researcher immediately saw that her original query was inappropriate simply due to the frequencies

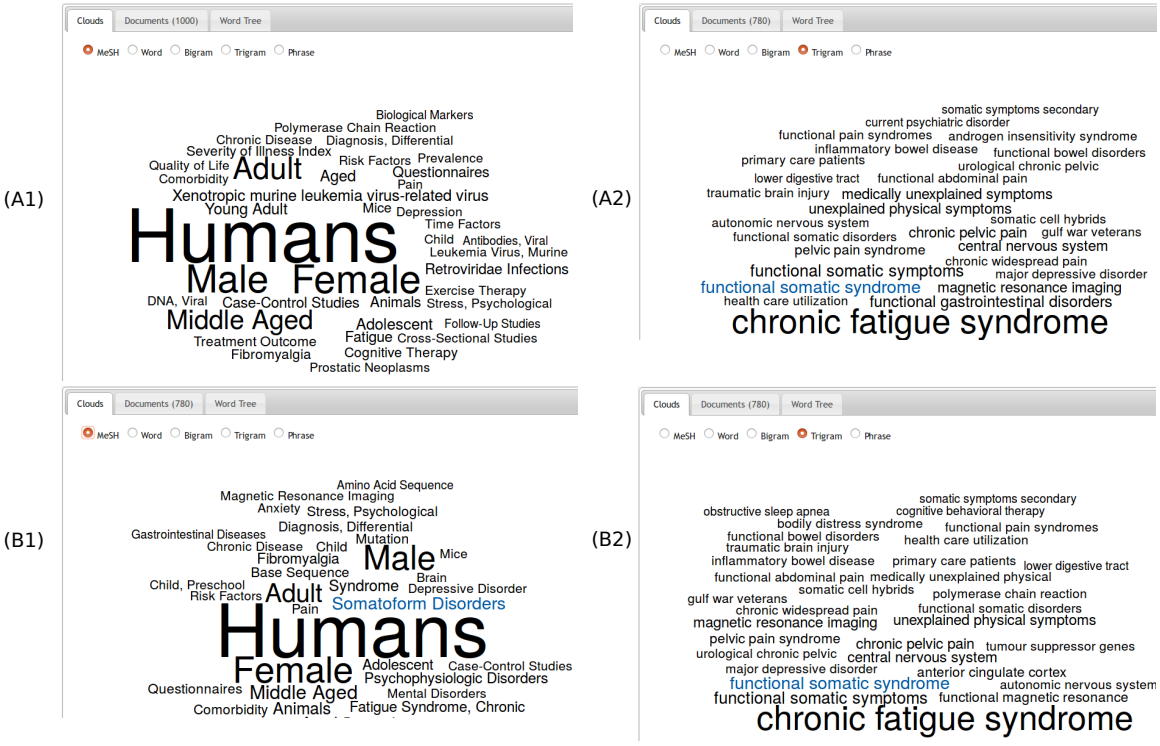


Figure 7.19: Differences between A1/A2 (MeSH and trigram clouds for *chronic fatigue syndrome*) and B1/B2 (MeSH and trigram clouds for *functional somatic syndromes*) illustrate the need for multiple lenses when searching; the MeSH cloud in A1 is missing the term *somatoform disorders* because it did not reach the minimum frequency required to be illustrated.

displayed by the clouds; it was determined that, by including the word *inflammation* inside a conjunction, the query results were being poisoned with results related to the human liver for a disease that typically has very little to do with the liver. The refined and corrected query contained approximately twenty percent fewer documents and was ultimately deemed a more successful query by the researcher. In Figure 7.20, we give an example on how feedback is supplied back to the user. Recall that alongside the visualizations discussed, a list of documents is displayed showing what documents remain post filtering. The text of the abstract is shown with focused words highlighted; we discuss focusing more in Section 7.9.1. The goal is to help the researcher understand why this article was included in the results; if the article is not appropriate, the user can modify their query to attempt to calibrate the results. The overarching goal is not to create a perfect information retrieval system that ranks and suggests articles to researchers, but rather to enable the researcher to immerse themselves in an environment where they are free to explore, hypothesize, and understand

the context of the visualizations that surround them.

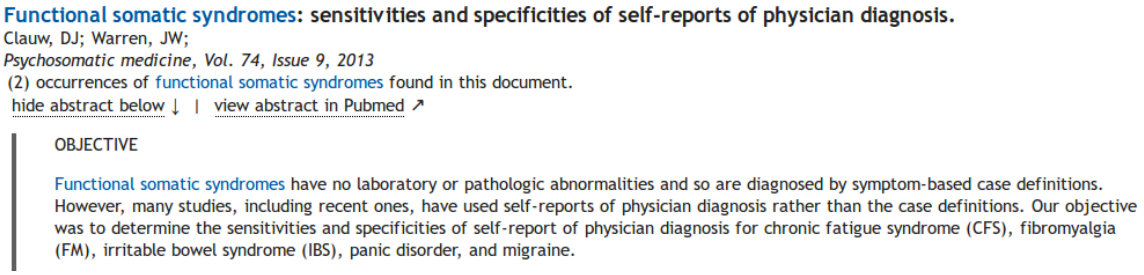


Figure 7.20: Terms and phrases can be selected as a point of focus; feedback such as highlighting the focused term or phrase must be given to the user so that they understand why this article was correctly or incorrectly included in the search results.

Scenario: Term Unfamiliarity

Similarly, a researcher may struggle with unfamiliarity of the terms used by an alternative discipline within an interdisciplinary collaboration. A muscle biologist was able to successfully reconcile differences between his own terminology and that of a bio-mechanical engineer by being presented with both frequencies and the associated context given by word trees. We also wish to explore how researcher preference and technical background plays a role in seeking information; because the interface provides a variety of analytic windows on top of the raw data, user choice becomes a pivotal element in whether or not researchers are successful in finding the information they seek.

7.9.1 Understanding DELVE

DELVE provides other collections of terms as facets for two reasons: 1) interdisciplinary collaboration typically involves researchers interested in biomedical literature who are not familiar with MeSH terms and 2) granularity and phrasing of terms can be an issue. From a modeling perspective, there are natural differences in the structure of the MeSH hierarchy and the collection of anchoring trigrams, but our categorical model naturally accounts for this by allowing objects to have any inclusive relationship within **Facet** categories: including those who have many relationships (MeSH terms) and those who have none (DELVE's trigrams). In DELVE's case, instances of facets play a role when creating focused collections of documents based on

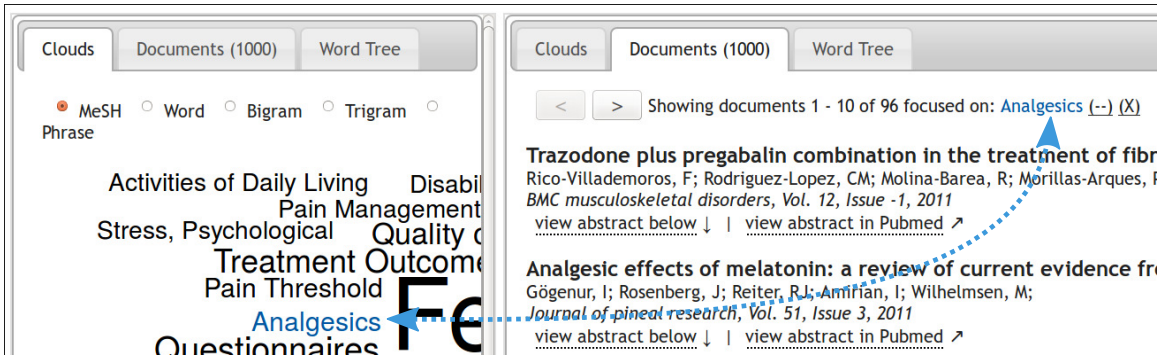


Figure 7.21: A DELVE search for fibromyalgia publications focusing on analgesics

what the user has selected through the interface, which could potentially span one or more facets.

Other visualizations, such as word trees and histograms, are available as part of the extensible nature of DELVE. We give an example of MeSH clouds and word trees working together in Section 7.9.2.

Focusing Considerations

The annotated screen-shot in Figure 7.21 demonstrates DELVE's ability to use a facet to focus. In this example, a search for fibromyalgia is focused on the MeSH term *analgesics*, which causes the documents viewer to show only those documents that are classified as belonging to the MeSH term *analgesics*. Multiple points of focus are supported in the subsequent version of DELVE, such as focusing using different word clouds [70] and word trees [76]. If the user also selects the MeSH term *female*, the document viewer would only show those documents tagged with both MeSH terms *analgesics* and *female*. Color is used to visually offset the facets being focused upon. The document viewer ranks results according to how many occurrences of the focus terms can be found within the abstract of the corresponding article.

Within one faceted taxonomy, aggregating focuses becomes a focused version of the queries discussed in Chapter 2. If we have created instances of **Facet** categories as discussed in Section 5.1.1, we can also create instances of focused subcategories by taking a subgraph of the graph underlying **Facet**:

Definition 23. *Given instances* $I_0, I_1, \ldots, I_N$ *of categories* $\boldsymbol{Facet}_0, \ldots, \boldsymbol{Facet}_N$*, let* $I_{F0}, I_{F1}, \ldots, I_{FN}$ *be focused instances created by replacing* $U((\boldsymbol{Facet}_i))$ *with* $U(\boldsymbol{Focus}_i)$ *for* $i = 0, 1, \ldots, N$*.*

Equation 3.7.1 indicated how to pinpoint objects of focused instances by taking the disjoint union of the inclusions necessary to create the focused collection of objects; Definition 23 simplifies this notation by structuring instances of focused facets as graphs where the objects can be referenced by definition simply due to our representation of the graph as a tuple.

7.9.2 Interacting with Word Clouds and Trees

In DELVE, facets contained in visualizations can work together harmoniously through a centralized point of focus; by default, focusing in one visualization will set the focus in all other visualizations. In Figure 7.22, we show a DELVE search for fibromyalgia and the result of focusing on the MeSH term *depression*. Within the MeSH cloud, the term is highlighted with blue and a secondary reminder cue containing the focused term is placed below the original query. The word tree redraws itself with the selected term as the root of the tree; this shows occurrences of the term *depression* within the sentences belonging to the classified resources, where redundancy is collapsed to a common prefix. For example, the following phrases are collapsed under tree nodes for *depression* followed by *anxiety*:

1. *depression, anxiety, and headache.*
2. *depression, anxiety, poor sleep quality and poor physical fitness...*
3. *depression, anxiety, muscle pain, autoimmune and thyroid disease...*

From Figure 7.22, we can also see that the phrases of the form {*depression, anxiety, and ...*} and the phrase {*depression, but not with anxiety*} point to different resources containing relationships between *fibromyalgia*, *depression*, and *anxiety*. The goal of DELVE is to immerse a researcher into an exploratory search system where visualizations help expedite the discovery process. This goal is made easier by constructing DELVE upon a solid theoretical foundation that has been demonstrated to intelligently reuse and integrate existing biomedical terminologies.

7.10 Future DELVE Considerations

In this chapter, we presented DELVE, a framework for developing interactive visualizations as modular Web-applications; we targeted biomedical publications available via Pubmed to assist researchers with exploratory search for research. We demonstrated that DELVE is a special case of the category-theoretic model of faceted brows-

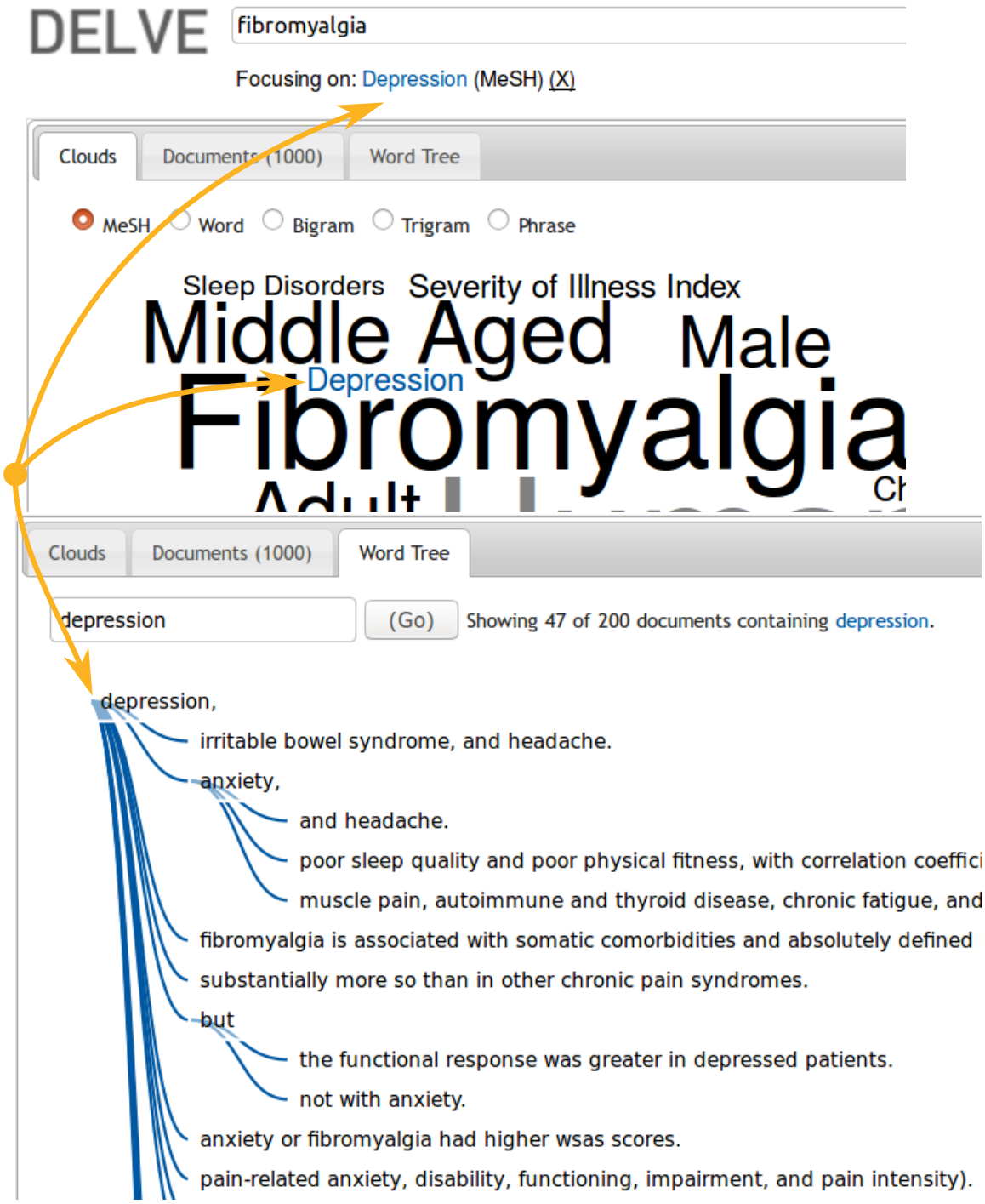


Figure 7.22: A DELVE search for fibromyalgia publications focusing on depression

ing; in this case, visualizations both contain facets and are driven by facets. This abstract framework enables the consistent design and implementation of reusable and interoperable DELVE applications. We also presented a publicly available prototype that demonstrates and integrates several DELVE-based visualizations. Preliminary evaluation indicated that DELVE was helpful in tuning a researcher's query for appropriateness and for helping cross barriers in interdisciplinary research by providing access to multiple lexicons understood by opposing fields. We are working to expand our library of DELVE applications in order to provide a complete suite of interoperable visualizations that immerse a researcher into an environment where research needs are easily serviced. Additional visualizations could incorporate the results of data mining and machine learning results upon the resources to be explored. If the computed results are mappable to a taxonomy or produce output according to an existing vocabulary or data standard, they can be easily integrated into our model.

Chapter 8 Conclusions

We contribute a model of faceted browsing that uses category-theory as its theoretical foundation; our goal is to provide a common language and environment that can unify existing and future research or development efforts with respect to faceted browsing. We wish to support a research environment that emphasizes reuse and interoperability in order for the propagation of novel components rather than their isolation to a single implementation.

8.1 Future Work

We are applying our model to the next stage of our interface and framework, DELVE [8], in order to represent faceted structures that help create or control other faceted structures. Together with deeper elements of category theory, our model will help inform how to build a proper application programming interface (API) for faceted browsing. The mapping between schemas and facets clears the path to implementation using a database containing tables for faceted data and faceted taxonomies. Support for functional databases is growing [29, 30], but a traditional relational database is adequate to store faceted relationships. An API for faceted browsing can bridge the gap between a categorical model for faceted browsing and databases, allowing us to start with traditional relational databases and migrate toward functional databases as they mature and become a realistic option for web-driven and data-driven applications.

We have demonstrated that it is possible to represent facets, faceted taxonomies, and faceted queries with category theory. Once the faceted query is performed, the interface must allow the user to successfully interact and engage with the faceted information being presented. We wish to continue our model to include this exploratory search phase of faceted browsing by leveraging deeper elements of category theory. For example, if an interface was designed to fulfill Shneiderman's information seeking mantra [69], how can tasks such as overview, filter, zoom, and details-on-demand be modeled? How can additional tasks from Shneiderman's task taxonomy [69], such as seeing relations, history, and extraction, be modeled?

Furthermore, if each of these tasks or interactions can be abstracted, does this model suggest anything about being able to measure the usability of the system? Usability is most commonly measured qualitatively, but we would like to explore the

possibility of quantitatively measuring usability.

As mentioned previously, a natural consequence of modeling facets, faceted taxonomies, and faceted browsing systems is that resources ultimately get retrieved. This opens the door to abstractly modeling and developing categorical manipulations of faceted data in a way that is transparent and reusable across systems. For example, we demonstrated that the categorical constructions pullbacks and pushouts can help easily organize and reorganize faceted data, resulting in new constructions such as faceted views that house interactive facets that do not directly exist in the source taxonomy. These and other types of related operations could potentially lead to creating facets dynamically, where new facets are created in real-time from computations involving existing ones potentially as a consequence of interactivity. Other operations, such as retractions, need to be explored so that their role in the model is fully understood; this is the groundwork toward the next steps of ranking and sorting resources.

The impact that visualizations play in faceted browsing systems deserves to be explored further. In systems such as DELVE, one interaction can have consequences in many parts of the interface. Ultimately, with a category-theoretic model, one will be able to prove something is mathematically possible before implementation; the relationships and road maps between proof and implementation paths need to be researched further.

8.2 Conclusions

We have demonstrated that category theory can act as a theoretical foundation for faceted browsing that also encourages reuse and interoperability. We have established facets and faceted taxonomies as categories and have demonstrated how the computational elements of category theory, such as products, functors, pushouts, and pullbacks, extend the usefulness of our model.

The utility of faceted browsing systems is well-established in the digital libraries research community [6, 7], but current efforts would benefit from a more abstract framework that encourages reuse and interoperability. In this context, reuse and interoperability are at two levels: between systems and within a system. Our model works at both levels by leveraging category theory as a common language for representation and computation. Without this common language, it is difficult to abstractly model a system that utilizes multiple faceted structures (hierarchies, trees, graphs, lattices), even if there are shared notations and definitions between them.

We have demonstrated that category theory can be used to model faceted browsing and that it offers a consistent view of facets as objects and morphisms between objects. With our general framework for communicating mathematically about facets at a high level of abstraction, we can construct interoperable interfaces and reuse existing efforts intelligently.

Appendix A: Examples of Facet Models

It can be difficult to understand the nature of our proposed category-theoretic model of facets without seeing other examples of facet models. In this section, we review in detail two well-known facet models that have a fundamentally different theoretical foundation than our proposed model.

Recall these two facet modeling efforts that use set theory as a foundation: FaSet [25] and Category Hierarchies [23]. They differ in their key definitions and how they model filtering and ranking. This example will outline each effort in detail and describe how each model defines and interacts with facets.

A Review of FaSet

In FaSet [25], Bonino et al. defines a facet as "an independent point of view for representing the content of a resource"; key definitions from the paper:

1. A facet F is a set of items; in systems with multiple facets, they are disjoint: $F_a \cap F_b = \emptyset$. The items in the set represent labels for that facet.

2. The facet space U is the universe set defined by the Cartesian product of all facets: $U = F_a \times F_b \times F_c \times \ldots$

3. A focus L is a named subset of F : $L \subseteq F$, where the name is a nullable, variable-length list of indexes: $L\langle i, j, k, \ldots \rangle$.

4. The classification of a resource r is the subset of F that is relevant, denoted as $r \perp F$. A sharp classification is a classification, for some set of focus names P, that can be expressed as a union of foci: $\exists P : r \perp F = \bigcup_{p \in P} L\langle p \rangle$

5. The multi-dimensional, sharp classification of a resource r with respect to a facet space U is defined as a product: $r \perp F = (r \perp F_a) \times (r \perp F_b) \times (r \perp F_c) \times \ldots$, which given the definition of a sharp classification also implies $r \perp F = \bigcup_{p \in P_a} L_a\langle p \rangle \times \bigcup_{p \in P_b} L_b\langle p \rangle \times \ldots$

This allows sharp classifications to be written as lists, which is easily implementable.

Dimensions of Facet Types

FaSet supports flat, hierarchical, and nested facet types. Nobel Prize category is the example given for a flat facet type, having values such as chemistry and physics. Geographical region is the example given for hierarchical: one can select country, state, city, etc. Age ratings are given as an example of a nested facet type, where if a video game is appropriate for a 7+ years old child, then it is also appropriate for a 12+ years one.

For flat and hierarchical facet types, multiple values can be focused upon. Language is an example of a flat facet that allows multiple focuses; for example, one can indicate they speak both English and Spanish. Research paper topics are an example given for a hierarchical facet type that allows multiple values: for example, one could indicate a paper covers both user models and user studies, which are underneath human-computer interaction.

For example, for facet F_c, $P_c = \{\langle 1\rangle, \langle 3\rangle, \langle 5\rangle\}$, is a simple flat-type of facet that allows multiple foci. This can be contrasted with for facet F_e, $P_e = \{\langle 2\rangle, \langle 3, 2, 1\rangle\}$, which is a hierarchical-type facet that allows multiple foci.

Filtering and Ranking

Bonino defines filtering as a stateless process for determining which resources are compatible with a query. Compatibility is formally defined for a given query q, a resource r, and a classification U, as $C = (q \perp U) \cap (r \perp U) \neq \emptyset$. This definition is used to construct a definition for prefix compatibility of two foci, denoted as $L_a\langle p_1\rangle \asymp L_a\langle p_2\rangle$, where p_1 and p_2 are either equivalent or prefixes of the other. Filtering can formally defined as $\exists p_q \in P_a(q), \exists p_r \in P_a(r) : L_a\langle p_q\rangle \asymp L_a\langle p_r\rangle$. In other words, filtering simply becomes checking the prefix compatibility of the facets belonging to the query and resource.

Bonino defines ranking as an ordering of the faceted resources that were compatible with the query; the general idea is that if resources have facets and the query is a faceted query, then one can measure the similarity between query and resource.

To formally construct a definition of ranking, Bonino gives:

1. The depth $D(L_a\langle p\rangle)$ of a focus $L_a\langle p\rangle$ is the number of hierarchy levels that compose the name of the focus.

2. Given two foci $L_a\langle p_1\rangle$ and $L_a\langle p_2\rangle$ of a same facet F_a, they define their focus

similarity as:

$$S(L_a\langle p_1\rangle, L_b\langle p_2\rangle) = W(\min\{D(L_a\langle p_1\rangle), D(L_a\langle p_2\rangle)\})$$

By definition, this is for when $L_a\langle p_1\rangle \asymp L_a\langle p_2\rangle$ and is 0 otherwise. W is a vector of increasing weights that map the depth level to a real number on the interval $(0, 1]$: these are definable as necessary. The example uses $W(0) = .10$, $W(1) = .33$, $W(2) = .45$, $W(3) = .60$, and $W(1) = .80$.

3. The focus similarities can be rolled up per each facet F_a for a given resource r and query q:

$$S_a(q, r) = \bigoplus_{p_q \in (q \perp F_a)} [\max_{P \in (r \perp F_a) \wedge p_q \asymp p_r} S(L_a\langle p_q\rangle, L_a\langle p_r\rangle)]$$

where the algebraic sum (or "probabilistic OR") is defined as $a \oplus b = a + b - a \cdot b$; this also continues to restrict the solution to the interval $[0, 1)$. Different facets have different depths and foci, so a normalized in-facet similarity is proposed so that similarity can be relative across facets: $S^*_q(q, r) = S_a(q, r)/S_a(q, q)$.

4. The authors intersect the similarity values with the "probabilistic AND" operator defined as $a \otimes b = a \cdot b$:

$$S(q, r) = \bigotimes_{a \in facets} S^*_q(p, r)$$

The resources can then be ranked according to their similarity measure.

The added benefit of this method of modeling facets is that it is almost directly implementable with SQL. The authors demonstrate how to represent the data in a relational database and how to build filtering and similarity operators.

A Review of Category Hierarchies

Facetedpedia is a faceted system built on top of Wikipedia, which demonstrates how to use directed acyclic graphs as the key data structure of a facet model[23]. The use of the word *category* in this model is unrelated to category theory.

Li defines a facet as dimensions/attributes/properties of objects and a faceted interface as a set of category hierarchies for a set of objects. Wikipedia offers category and sub-category relationships and Facetpedia automatically constructs a faceted hierarchy for a given query based on these relationships. Key definitions:

1. The target articles of a given query q is the set of top-s ranked Wikipedia articles $T = \{p_1, \ldots, p_s\}$. Each Wikipedia article $p\prime$ that is hyperlinked from a target article p is called an attribute article, represented as $p\prime \leftarrow p$. The set of attribute articles is $A = \{p\prime_1, \ldots, p\prime_m\}$.

2. A category hierarchy is a connected, rooted directed acyclic graph $H(r_H, C_H, E_H)$, where r_H is the root category, $C_H = \{c\}$ is the set of categories, and $E_H = \{c \rightarrow c\prime\}$ is the set of category to subcategory relationships.

3. A facet $F(r, C_F, E_F)$ is a rooted, connected subgraph of the category hierarchy $H(r_H, C_H, E_H)$, where $C_F \subseteq C_H$, $E_F \subseteq E_H$, and $r \in C_F$ is the root of F.

4. A facet is a safe reaching facet if $\forall c \in C_F$, there exists a target article $p \in T$ such that c reaches p.

5. A navigational path in F is a sequence $c_1 \rightarrow \ldots \rightarrow c_t \Rightarrow p\prime \leftarrow p$, where,

   a) for $1 \leq i \leq t$, $c_i \in C_F$

   b) for $1 \leq i \leq t-1$, $c_i \rightarrow c_{i+1} \in E_F$

   c) $p\prime \in A$, and c_t is a category of $p\prime$, represented as $c_t \Rightarrow p\prime$.

   d) $p \in T$, and $p\prime$ is an attribute article of p ($p \rightarrow p\prime$).

Bibliography

[1] B. Wei, J. Liu, Q. Zheng, W. Zhang, X. Fu, and B. Feng, "A survey of faceted search," *Journal of Web engineering*, vol. 12, no. 1-2, pp. 41–64, 2013.

[2] M. A. Hearst, "Design recommendations for hierarchical faceted search interfaces," in *SIGIR workshop on faceted search*. ACM, 2006, pp. 1–5.

[3] ——, "Clustering versus faceted categories for information exploration," *Communications of the ACM*, vol. 49, no. 4, pp. 59–61, 2006.

[4] D. R. Harris, "Modeling reusable and interoperable faceted browsing systems with category theory," in *Information Reuse and Integration (IRI), 2015 IEEE International Conference on*. IEEE, 2015, pp. 388–395.

[5] ——, "Foundations of reusable and interoperable facet models using category theory," *Information Systems Frontiers*, pp. 1–13, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10796-016-9658-6

[6] J. C. Fagan, "Usability studies of faceted browsing: a literature review," *Information Technology and Libraries*, vol. 29, no. 2, pp. 58–66, 2013.

[7] X. Niu and B. Hemminger, "Analyzing the interaction patterns in a faceted search interface," *Journal of the Association for Information Science and Technology*, 2014. [Online]. Available: http://dx.doi.org/10.1002/asi.23227

[8] D. R. Harris, R. Kavuluru, S. Yu, R. Theakston, J. W. Jaromczyk, and T. R. Johnson, "Delve: A document exploration and visualization engine," in *Proceedings of the Summit on Clinical Research Informatics*. AMIA, 2014, p. 179.

[9] K. P. Yee, K. Swearingen, K. Li, and M. Hearst, "Faceted metadata for image search and browsing," in *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 2003, pp. 401–408.

[10] D. I. Spivak, *Category Theory for the Sciences*. MIT Press, 2014.

[11] M. Barr and C. Wells, *Category theory for computing science*. Prentice Hall New York, 1990.

[12] B. C. Pierce, *Basic category theory for computer scientists*. MIT press, 1991.

[13] S. De Lusignan, N. Sadek, H. Mulnier, A. Tahir, D. Russell-Jones, and K. Khunti, "Miscoding, misclassification and misdiagnosis of diabetes in primary care," *Diabetic medicine*, vol. 29, no. 2, pp. 181–189, 2012.

[14] S. N. Murphy, G. Weber, M. Mendis, V. Gainer, H. C. Chueh, S. Churchill, and I. Kohane, "Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2)," *Journal of the American Medical Informatics Association*, vol. 17, no. 2, pp. 124–130, 2010.

[15] A. Dawson, D. Brown, and V. Broughton, "The need for a faceted classification as the basis of all methods of information retrieval," in *Aslib proceedings*, vol. 58, no. 1/2. Emerald Group Publishing Limited, 2006, pp. 49–72.

[16] D. Fuentes-Lorenzo, J. Morato, and J. M. Gómez, "Knowledge management in biomedical libraries: A semantic web approach," *Information Systems Frontiers*, vol. 11, no. 4, pp. 471–480, 2009.

[17] H.-J. Chu and R.-C. Chow, "An information model for managing domain knowledge via faceted taxonomies," in *Information Reuse and Integration (IRI), 2010 IEEE International Conference on*. IEEE, 2010, pp. 378–379.

[18] R. Prieto-Díaz, "A faceted approach to building ontologies," in *Information Reuse and Integration, 2003. IRI 2003. IEEE International Conference on*. IEEE, 2003, pp. 458–465.

[19] U. Priss, "Faceted information representation," in *Proceedings of the 8th International Conference on Conceptual Structures*. Springer-Verlag, 2000, pp. 84–94.

[20] F. Giunchiglia, M. Marchese, and I. Zaihrayeu, "Encoding classifications into lightweight ontologies," in *Journal on data semantics*. Springer, 2007, pp. 57–81.

[21] A. Blass, "The interaction between category theory and set theory," in *Mathematical applications of category theory*. American Mathematical Society, 1984, vol. 89, pp. 84–84.

[22] G. M. Sacco and Y. Tzitzikas, *Dynamic taxonomies and faceted search: theory, practice, and experience*. Springer, 2009.

[23] C. Li, N. Yan, S. B. Roy, L. Lisham, and G. Das, "Facetedpedia: dynamic generation of query-dependent faceted interfaces for wikipedia," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 651–660.

[24] E. C. Clarkson, S. B. Navathe, and J. D. Foley, "Generalized formal models for faceted user interfaces," in *Proceedings of the 9th ACM/IEEE Joint Conference on Digital libraries*. ACM, 2009, pp. 125–134.

[25] D. Bonino, F. Corno, and L. Farinetti, "Faset: A set theory model for faceted search," in *Proceedings of the IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*. IEEE, 2009, pp. 474–481.

[26] B. Coecke and É. O. Paquette, "Categories for the practising physicist," in *New Structures for Physics*. Springer, 2011, pp. 173–286.

[27] S. Phillips and W. H. Wilson, "Categorial compositionality: A category theory explanation for the systematicity of human cognition," *PLoS computational biology*, vol. 6, no. 7, p. e1000858, 2010.

[28] D. I. Spivak, T. Giesa, E. Wood, and M. J. Buehler, "Category theoretic analysis of hierarchical protein materials and social networks," *PLoS One*, vol. 6, no. 9, p. e23911, 2011.

[29] D. I. Spivak, "Simplicial databases," *arXiv preprint arXiv:0904.2012*, 2009.

[30] ——, "Functorial data migration," *Information and Computation*, vol. 217, pp. 31–51, 2012.

[31] D. I. Spivak and R. E. Kent, "Ologs: a categorical framework for knowledge representation," *PLoS One*, vol. 7, no. 1, p. e24274, 2012.

[32] P. Vickers, J. Faith, and N. Rossiter, "Understanding visualization: A formal approach using category theory and semiotics," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 19, no. 6, pp. 1048–1061, 2013.

[33] H. K. Wall, J. A. Hannan, and J. S. Wright, "Patients with undiagnosed hypertension: Hiding in plain sight," *JAMA*, vol. 312, no. 19, pp. 1973–1974, 2014.

[34] E. F. Kern, M. Maney, D. R. Miller, C.-L. Tseng, A. Tiwari, M. Rajan, D. Aron, and L. Pogach, "Failure of icd-9-cm codes to identify patients with comorbid

chronic kidney disease in diabetes," *Health services research*, vol. 41, no. 2, pp. 564–580, 2006.

[35] D. R. Harris, D. W. Henderson, and J. C. Talbert, "Using closure tables to enable cross-querying of ontologies in database-driven applications," in *Biomedical and Health Informatics (BHI), 2017 IEEE-EMBS International Conference on*. IEEE, 2017, pp. 1–4.

[36] R. A. Cote and S. Robboy, "Progress in medical information management: Systematized nomenclature of medicine (snomed)," *Jama*, vol. 243, no. 8, pp. 756–762, 1980.

[37] R. Cornet and N. de Keizer, "Forty years of snomed: a literature review," *BMC medical informatics and decision making*, vol. 8, no. 1, p. S2, 2008.

[38] B. Kules, R. Capra, M. Banta, and T. Sierra, "What do exploratory searchers look at in a faceted search interface?" in *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*. ACM, 2009, pp. 313–322.

[39] D. E. Rydeheard and R. M. Burstall, *Computational category theory*. Prentice Hall, 1988.

[40] M. Odersky, "The scala language specification, version 2.4," Programming Methods Laboratory, EPFL, Lausanne, Switzerland, Technical Report, 2007.

[41] H. Seeberger, "Introduction to category theory in scala," November 2010, retrieved January 25, 2015 from http://hseeberger.github.io/blog/2010/11/25/introduction-to-category-theory-in-scala/.

[42] L. Hupel, "scalaz: functional programming for scala," 2014, retrieved January 25, 2015 from http://typelevel.org/projects/scalaz/.

[43] ——, "scalaz.category api documentation," 2014, retrieved January 25, 2015 from http://docs.typelevel.org/api/scalaz/stable/7.1.0-M3/doc/#scalaz.Category.

[44] D. L. McGuinness, F. Van Harmelen *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[45] B. Karwin, *SQL antipatterns: avoiding the pitfalls of database programming*, 1st ed. Pragmatic Bookshelf, 2010.

[46] T. Bray, "The javascript object notation (json) data interchange format," March 2014, retrieved June 15, 2016 from https://tools.ietf.org/html/rfc7159/.

[47] D. R. Harris, "Modeling terminologies for reusability in faceted systems," *Advances in Intelligent Systems and Computing*, pp. 1–25, 2017.

[48] ——, "Modeling integration and reuse of heterogeneous terminologies in faceted browsing systems," in *Information Reuse and Integration (IRI), 2016 IEEE International Conference on*. IEEE, 2016, p. pp.

[49] W. H. Organization *et al.*, "The icd-10 classification of mental and behavioural disorders: clinical descriptions and diagnostic guidelines," in *The ICD-10 Classification of Mental and Behavioural Disorders: Clinical descriptions and diagnostic guidelines*. World Health Organization, 1992.

[50] O. Bodenreider, "The unified medical language system (umls): integrating biomedical terminology," *Nucleic acids research*, vol. 32, no. suppl 1, pp. D267–D270, 2004.

[51] C. for Medicare & Medicaid Services, *Healthcare Common Procedure Coding System (HCPCS)*. Centers for Medicare & Medicaid Services, 2003.

[52] D. R. Harris, R. Kavuluru, J. W. Jaromczyk, and T. R. Johnson, "Rapid and reusable text visualization and exploration development with delve," in *Proceedings of the Summit on Clinical Research Informatics*. AMIA, 2017, pp. 1–10.

[53] A. G. Fraser and F. D. Dunstan, "On the impossibility of being expert," *BMJ*, vol. 341, p. c6815, 2010.

[54] L. Hunter and K. B. Cohen, "Biomedical language processing: what's beyond pubmed?" *Molecular cell*, vol. 21, no. 5, pp. 589–594, 2006.

[55] R. I. Dogan, G. C. Murray, A. Névéol, and Z. Lu, "Understanding pubmed® user search behavior through log analysis," *Database*, vol. 2009, p. bap018, 2009.

[56] "Using pubmed," 2016. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed

[57] Z. Lu, "Pubmed and beyond: a survey of web tools for searching biomedical literature," *Database*, vol. 2011, p. baq036, 2011.

[58] M. Song, "Exploring concept graphs for biomedical literature mining," in *2015 International Conference on Big Data and Smart Computing (BIGCOMP)*. IEEE, 2015, pp. 103–110.

[59] K. Lee, W. Shin, B. Kim, S. Lee, Y. Choi, S. Kim, M. Jeon, A. C. Tan, and J. Kang, "Hipub: translating pubmed and pmc texts to networks for knowledge discovery," *Bioinformatics*, 2016. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/early/2016/08/20/bioinformatics.btw511.abstract

[60] Y. Zhang, I. N. Sarkar, and E. S. Chen, "Pubmedminer: mining and visualizing mesh-based associations in pubmed," in *AMIA Annual Symposium Proceedings*, vol. 2014. American Medical Informatics Association, 2014, p. 1990.

[61] S. Kim, L. Yeganova, and W. J. Wilbur, "Meshable: searching pubmed abstracts by utilizing mesh and mesh-derived topical terms," *Bioinformatics*, p. btw331, 2016.

[62] N. Médoc, M. Ghoniem, and M. Nadif, "Exploratory analysis of text collections through visualization and hybrid biclustering," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2016, pp. 59–62.

[63] P. Ernst, A. Siu, D. Milchevski, J. Hoffart, and G. Weikum, "Deeplife: An entity-aware search, analytics and exploration platform for health and life sciences," *ACL 2016*, p. 19, 2016.

[64] J.-D. Fekete, J. J. Van Wijk, J. T. Stasko, and C. North, "The value of information visualization," in *Information visualization*. Springer, 2008, pp. 1–18.

[65] K. Kucher and A. Kerren, "Text visualization techniques: Taxonomy, visual survey, and community insights," in *2015 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 2015, pp. 117–121.

[66] W. Dou, X. Wang, D. Skau, W. Ribarsky, and M. X. Zhou, "Leadline: Interactive visual analysis of text data through event identification and exploration," in *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*. IEEE, 2012, pp. 93–102.

[67] S. Koch, M. John, M. Wörner, A. Müller, and T. Ertl, "Varifocalreaderin-depth visual analysis of large text documents," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 1723–1732, 2014.

[68] F. B. Viegas, M. Wattenberg, F. Van Ham, J. Kriss, and M. McKeon, "Manyeyes: a site for visualization at internet scale," *IEEE transactions on visualization and computer graphics*, vol. 13, no. 6, pp. 1121–1128, 2007.

[69] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Visual Languages, 1996. Proceedings., IEEE Symposium on.* IEEE, 1996, pp. 336–343.

[70] F. B. Viegas, M. Wattenberg, and J. Feinberg, "Participatory visualization with wordle," *IEEE transactions on visualization and computer graphics*, vol. 15, no. 6, pp. 1137–1144, 2009.

[71] H. J. Lowe and G. O. Barnett, "Understanding and using the medical subject headings (mesh) vocabulary to perform literature searches," *Journal of the American Medical Association*, vol. 271, no. 14, pp. 1103–1108, 1994.

[72] "Django," 2016. [Online]. Available: https://www.djangoproject.com/

[73] "Delve," 2016. [Online]. Available: https://bitbucket.org/_harris/delve

[74] D. Crockford, "The application/json media type for javascript object notation (json)," 2006.

[75] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE transactions on visualization and computer graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.

[76] M. Wattenberg and F. B. Viégas, "The word tree, an interactive visual concordance," *IEEE transactions on visualization and computer graphics*, vol. 14, no. 6, pp. 1221–1228, 2008.

[77] F. Van Ham, M. Wattenberg, and F. B. Viégas, "Mapping text with phrase nets." *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 6, pp. 1169–1176, 2009.

[78] J. Harris, "Word clouds considered harmful," *Nieman Journalism Lab*, 2011.

[79] I. N. Sarkar, R. Schenk, H. Miller, and C. N. Norton, "Ligercat: using "mesh clouds" from journal, article, or gene citations to facilitate the identification of relevant biomedical literature." American Medical Informatics Association, 2009.

[80] C. Baroukh, S. L. Jenkins, R. Dannenfelser, and A. Ma'ayan, "Genes2wordcloud: a quick way to identify biological themes from gene lists and free text," *Source code for biology and medicine*, vol. 6, no. 1, p. 1, 2011.

[81] "Mesh browser," 2017. [Online]. Available: https://meshb.nlm.nih.gov

[82] A. Spink, D. Wolfram, M. B. Jansen, and T. Saracevic, "Searching the web: The public and their queries," *Journal of the American society for information science and technology*, vol. 52, no. 3, pp. 226–234, 2001.

Vita

Name

Daniel R. Harris

Education

- B.S., Computer Science and Mathematics, University of Kentucky, 2008.

Work Experience

- *Mar. 2016– current:* Application Systems Architect. Institute for Pharmaceutical Outcomes and Policy and the Center for Clinical and Translational Sciences, University of Kentucky, Lexington, KY.
- *Aug. 2011– 2016:* Information Technology Manager II. Center for Clinical and Translational Sciences, University of Kentucky, Lexington, KY.
- *2011:* Systems Programmer. Institute for Pharmaceutical Outcomes and Policy, University of Kentucky, Lexington, KY.
- *2010– 2015:* Adjunct Instructor. University of Kentucky, Lexington, KY.
- *2009– 2010:* Research Assistant - High-performance Computing Support, University of Kentucky, KY.
- *2008– 2009:* Teaching Assistant/Instructor, University of Kentucky, Lexington, KY.
- *2007– 2008:* Undergraduate Assistant, University of Kentucky, Lexington, KY.
- *2006– 2007:* Internship/Java Developer. ArchVision, Lexington, KY.
- *2001– current:* Linux Server Administration. Personal and small business projects, Lexington, KY.

Honors

- First place at the 2017 IEEE Biomedical and Health Informatics Big Data Analytics and Visualization Competition.
- Third place presentation at the Kentucky Academy of Sciences 99th Annual Meeting.
- Selected as a Silver Medallion Poster (one of four awarded) by Dr. W. Ed Hammond at the 2013 AMIA Joint Summits on Translational Science.
- Nominated for a distinguished poster at the 2012 AMIA Annual Symposium.
- Recipient of the 2011 Thaddeus B. Curtz Memorial Award from the Department of Computer Science, University of Kentucky
- Second place presentation at the 2010 Eastern Kentucky Symposium in the Mathematical, Statistical, and Computer Sciences.

Certifications

- Risk-based De-identification of Health Data. Privacy Analytics. 2015.
- Big Data and Hadoop. Simplilearn. 2014.

Associated Selected Publications

This dissertation is largely an expanded and integrated version of the following conference submissions and journal articles:

1. **Daniel R. Harris** " Modeling Terminologies for Reusability in Faceted Systems." Advances in Intelligent Systems and Computing, pp 1-25. Springer, 2017.
2. **Daniel R. Harris**, Ramakanth Kavuluru, Jerzy W. Jaromczyk, Todd R. Johnson. "Rapid and Reusable Text Visualization and Exploration Development with DELVE". In Proceedings of the 2017 American Informatics Association (AMIA) Joint Summits (CRI), pp. 1-10. AMIA, 2017.
3. **Daniel R. Harris** "Modeling integration and reuse of heterogeneous terminologies in faceted systems." In Proceedings of the 2016 IEEE International

Conference on Information Reuse and Integration (IRI), pp. 58-66. IEEE, 2016. https://doi.org/10.1109/IRI.2016.16

4. **Daniel R. Harris** "Foundations of reusable and interoperable facet models using category theory." Information Systems Frontiers, vol. 18, no. 5, pp. 953-965. Springer, 2016. https://doi.org/10.1007/s10796-016-9658-6

5. **Daniel R. Harris** "Modeling reusable and interoperable faceted browsing systems with category theory." In Proceedings of the 2015 IEEE International Conference on Information Reuse and Integration (IRI), pp. 388-395. IEEE, 2015. https://doi.org/10.1109/IRI.2015.65

6. **Daniel R. Harris** "Modeling faceted browsing with category theory to support interoperability and reuse." In Proceedings of the 15th ACM/IEEE-CE on Joint Conference on Digital Libraries, pp. 275-276. ACM, 2015. https://doi.org/10.1145/2756406.2756972

Other Publications

1. **Daniel R. Harris**, Darren W. Henderson, Jeffery C. Talbert. "Using Closure Tables to Enable Cross-Querying of Ontologies in Database-Driven Applications." In Proceedings of the 2017 IEEE International Conference on Biomedical and Health Informatics (BHI), pp. 1-4. IEEE, 2017.

2. **Daniel R. Harris**, Adam D. Baus, Tamela J. Harper, Traci D. Jarrett, Cecil R. Pollard, Jeffery C. Talbert. "Using i2b2 to bootstrap rural health analytics and learning networks." In Proceedings of the 2016 IEEE 38th Annual International Conference of the Engineering in Medicine and Biology Society (EMBC), pp. 2533-2536. IEEE, 2016. https://doi.org/10.1109/EMBC.2016.7591246

3. **Daniel R. Harris**, Darren W. Henderson. "i2b2t2: Unlocking Visualization for Clinical Research." In Proceedings of the 2016 American Informatics Association (AMIA) Joint Summits (CRI), pp.98-104. AMIA, 2016. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5001764/

4. **Daniel R. Harris**, Tamela J. Harper, Darren W. Henderson, Keith W. Henry, Jeffery C. Talbert. "Informatics-based Challenges of Building Collaborative Healthcare Research and Analysis Networks from Rural Community Health

Centers." In Proceedings of the 2016 IEEE International Conference on Biomedical and Health Informatics (BHI), pp. 513-516. IEEE, 2016. https://doi.org/10.1109/BHI.2016.7455947

5. **Daniel R. Harris**, Darren W. Henderson, Ramakanth Kavuluru, Arnold Stromberg, Todd R. Johnson. "Using Common Table Expressions to Build a Scalable Boolean Query Generator for Clinical Data Warehouses." IEEE Journal of Biomedical and Health Informatics, vol. 18, no. 5, pp. 1607-1613. IEEE, 2014. https://doi.org/10.1109/JBHI.2013.2292591

6. Ramakanth Kavuluru, Sifei Han, **Daniel R. Harris**. "Unsupervised Extraction of Diagnosis Codes from EMRs Using Knowledge-Based and Extractive Text Summarization Techniques." Advances in Artificial Intelligence, pp. 77-88. Springer, 2013. https://doi.org/10.1007/978-3-642-38457-8_7

7. Ramakanth Kavuluru, **Daniel R. Harris**. "A Knowledge-Based Approach to Syntactic Disambiguation of Biomedical Noun Compounds." In Proceedings of the 24th International Conference on Computational Linguistics (COLING Posters) pp. 559-568. ACL, 2012. http://www.aclweb.org/anthology/C12-2055

Posters

- **DELVE: A Document Exploration and Visualization Engine.** Daniel R. Harris, Ramakanth Kavuluru, Zhiguo Yu, Robert H. Theakston, Jerzy W. Jaromczyk, Todd R. Johnson. The 2014 AMIA Joint Summits on Translational Science.

- **Role of Named Entity Recognition in Extraction of Diagnosis Codes from Electronic Medical Records.** Daniel R. Harris, Todd R. Johnson, and Ramakanth Kavuluru. The 2013 AMIA Joint Summits on Translational Science. *Silver Medallion Poster.*

- **Improving Scalability and Performance of i2b2 Query Processing Using Common Table Expressions.** Darren W. Henderson, Daniel R. Harris, Ramakanth Kavuluru, Todd R. Johnson. The 2012 AMIA Annual Symposium. *Nominated for a distinguished poster.* Also presented at the 2012 CTSA Annual Meeting.

- **Bioinformatics Support for Collaborative Research.** Daniel R. Harris, Elisaveta Arnaudova, Hannah Atkinson, Tommy Bullock, Zhenmin Lin, Neil Moore, Christopher L. Schardl, Jerzy W. Jaromczyk. 97th Annual KY Academy of Science Meeting.

- **Moving from the HS21 Blade Cluster to the Lipscomb HPC Cluster for Bioinformatic Applications: A Case Study for InterproScan and BLAST.** Daniel R. Harris, Christopher L. Schardl, Jerzy W. Jaromczyk. 96th Annual KY Academy of Science Meeting.

- **Novel Deployment Methods and Modifications of InterproScan**. Daniel R. Harris, Christopher L. Schardl, Jerzy W. Jaromczyk. 15th Annual KY EPSCoR Conference.

- **Experimenting with Database Segmentation Size vs. Time Performance for mpiBLAST on an IBM HS21 Blade Cluster**. Daniel R. Harris, Christopher L. Schardl, Jerzy W. Jaromczyk. UT-ORNL-KBRIN Bioinformatics Summit 2010.

Presentations

- **Patient Cohort Discovery with i2b2 and Trinetx.** The 2017 CCTS Biomedical Informatics Core Series.

- **Augmenting Data with Semantics for Visualization.** The 2017 IEEE Big Data Analytics Competition. (1st Place Winner).

- **Cohort Discovery and Hypothesis Exploration.** The 2016 CCTS PSMRF Seminar Series.

- **Essentials of Linux.** 2015 Next Generation Sequencing and Data Analysis Workshop.

- **Essentials of Linux.** 2014 Next Generation Sequencing and Data Analysis Workshop.

- **DELVE: Document ExpLoration and Visualization Engine**. The University of Kentucky's Department of Computer Science Keeping Current Seminar.

- **An Introduction to DELVE: Document ExpLoration and Visualization Engine**. 99th Kentucky Academy of Science.
- **Essentials of Linux.** 2013 Next Generation Sequencing and Data Analysis Workshop.
- **Utilizing bioinformatics: tools and applications.** Guest lecture for CPH738-010: Tools and Applications in Biomedicine, University of Kentucky.
- **Essentials of Linux.** 2012 Next Generation Sequencing and Data Analysis Workshop.
- **What is DNA?** Interactive presentation for the 2012 University of Kentucky Engineering Day.
- **Experimenting with Database Segmentation Size vs. Time Performance for mpiBLAST on an IBM HS21 Blade Cluster**. The 2010 Eastern Kentucky Symposium in the Mathematical, Statistical, and Computer Sciences. (2nd place presentation.)
- **What is Bioinformatics?** Interactive presentation for the 2010 University of Kentucky Engineering Day.

Collaborative Efforts

- **Host tissue environment directs activities of an Epichloë endophyte, while it induces systemic hormone and defense responses in its native perennial ryegrass host.** Jan Schmid, Robert Day, Ningxin Zhang, Pierre-Yves Dupont, Murray P Cox, Christopher Lewis Schardl, Niki Minards, Mauro Truglio, Neil Moore, **Daniel R Harris**, Yanfei Zhou. Molecular Plant-Microbe Interactions, 2016 `http://dx.doi.org/10.1094/MPMI-10-16-0215-R`.
- **Plant-symbiotic fungi as chemical engineers: multi-genome analysis of the Clavicipitaceae reveals dynamics of alkaloid loci.** Christopher L Schardl, Carolyn A Young, Uljana Hesse, Stefan G Amyotte, Kalina Andreeva, Patrick J Calie, Damien J Fleetwood, David C Haws, Neil Moore, Birgitt Oeser, Daniel G Panaccione, Kathryn K Schweri, Christine R Voisey, Mark L Farman, Jerzy W Jaromczyk, Bruce A Roe, Donal M O'Sullivan, Barry Scott, Paul Tudzynski, Zhiqiang An, Elissaveta G Arnaoudova, Charles T Bullock, Nikki D Charlton, Li Chen, Murray Cox, Randy D Dinkins, Simona Florea,

Anthony E Glenn, Anna Gordon, Ulrich Gldener, **Daniel R Harris**, Walter Hollin, Jolanta Jaromczyk, Richard D Johnson, Anar K Khan, Eckhard Leistner, Adrian Leuchtmann, Chunjie Li, JinGe Liu, Jinze Liu, Miao Liu, Wade Mace, Caroline Machado, Padmaja Nagabhyru, Juan Pan, Jan Schmid, Koya Sugawara, Ulrike Steiner, Johanna E Takach, Eiji Tanaka, Jennifer S Webb, Ella V Wilson, Jennifer L Wiseman, Ruriko Yoshida, Zheng Zeng. PLoS genetics 9 (2), e1003323. `http://dx.doi.org/10.1371/journal.pgen.1003323`

- **Genome sequence of Epichloe festucae.** CL Schardl, N Moore, PX Zhao, EG Arnaoudova, CT Bullock, X Dai, **DR Harris**, J Jaromczyk, AK Khan, M Liu, LG Robinson, J Schmid, JS Webb, JL Wiseman, Z Zeng, ML Farman, U Hesse, JW Jaromczyk, J Liu, BA Roe, B Scott, CA Young. *Epichloe, endophytes of cool season grasses: implications, utilization and biology.* Proceedings of the 7th International Symposium on Fungal Endophytes of Grasses, Lexington, Kentucky, USA, 28 June to 1 July 2010 2012 pp. 59-64. ISBN 978-0-9754303-6-1.